Sentinel Mining

Middelfart, Morten

Publication date:
2010

Document Version
Accepted author manuscript, peer reviewed version

Sentinel Mining

Morten Middelfart

Ph.D. Dissertation

A dissertation submitted to the Faculty of Engineering and Science at Aalborg University, Denmark, in partial fulfillment of the requirements for the Ph.D. degree in computer science.

Abstract

This thesis introduces the novel concept of sentinel rules (sentinels). Sentinels are intended to represent the relationships between the data originating from the external environment and the data representing the critical organizational performance. The intention with sentinels is to warn business users about potential changes to Key Performance Indicators (KPIs) and thereby facilitate corrective action before such a change becomes a reality. In theory, sentinels have been described in the Computer Aided Leadership & Management (CALM) philosophy as a way to expand the window of opportunity for organizations, and thus a way facilitate successful navigation even though the world behaves chaotically due to globalization of commerce and connectivity. In this thesis, the sentinels from theory are turned into a reality such that common users at all organizational levels can benefit from the early warnings of sentinels.

Specifically, sentinels are rule relationships at the *schema level* in a multidimensional data cube. These relationships represent *changes over time* in certain measures that are followed by a change in a user defined critical measure, typically a KPI. An important property of a sentinel is *bi-directionality*, which means that the change relationship holds in the complement direction, since a sentinel with the bi-directional property has a higher chance of being causal rather than coincidental. Sentinels can vary in complexity depending on the number of measures that are included in the rule: *Regular sentinels* represent relationships where changes in one measure leads to changes in another within a given time frame. *Generalized sentinels* represent relationships between changes in multiple measures leading to changes in a given measure within a given time frame. *Multidimensional sentinels* combine the schema and the data levels, meaning that each measure change in the rule can hold for either *subsets* or the entire cube. A generalized sentinel could for example notify users that revenue might drop within two months if an increase in customer problems combined with a decrease in website traffic is observed, whereas a multidimensional sentinel could warn users that revenue might drop within two months if an increase in customer complaints in USA (drilldown into geography dimension) combined with a

decrease in the money invested in customer support for laptop computers (drilldown into product dimension) is observed.

The work leading to this thesis progressed from algorithms for regular sentinel mining with only one source and one target measure, into algorithms for mining generalized and multidimensional sentinels with multiple source measures. Furthermore, the mining algorithms became capable of automatically fitting the best warning periods for a given sentinel. Aside from expanding the capabilities of the algorithms, the work demonstrates a significant progression in the efficiency of sentinel mining, where the latest bitmap-based algorithms, that also take advantage of modern CPUs, are 3–4 orders of magnitude faster than the first SQL-based sentinel mining algorithm. This work also led to the industrial implementation of sentinel mining in the commercial software TARGIT BI Suite, which attracted the attention of leading industry analysts. In short, the work in this thesis has turned sentinel mining from a theoretical idea into concrete, highly efficient algorithms, and in addition it has demonstrated sentinels to be useful and unique.

Acknowledgments

There are many people I would like to thank for their help and support during my Ph.D. project. First of all, my thanks go to my Ph.D. supervisor, Torben Bach Pedersen, for his great inspiration and support of my work during the entire Ph.D. project. Aside from inspiration, Torben demonstrated how thorough and goal oriented research is practiced with excellence within the field of computer science.

Additional thanks go to Jan Krogsgaard and the rest of my colleagues in TARGIT A/S for their support during the project. Moreover, I would like to thank them for their supporting faith in the directions chosen with this research. In particular, the support of my CEO, Morten Sandlykke, was invaluable during the entire project.

Last, but definitely not least, I would like to thank my dear wife, Ginette, as well as Sophie and Louie for their patience and encouragement during the project; especially during the periods where deadlines needed to be met. A loving home was perhaps the strongest asset in completing this voyage.

This work was supported by TARGIT A/S, Cassiopeia Innovation and the European Regional Development Fund.

Contents

Chapter 1

Introduction

1.1 Motivation

On-Line Analytical Processing (OLAP) has been successfully applied to improve decision making in organizations, and as data volumes and update frequencies increase, there is a growing need for automated processes that rapidly capture the essence of the data. However, as globalized trading and connectivity increases, the pace and unpredictability of business operations increase as well. This means that traditional methods of user driven data discovery will be too slow, and traditional methods of long-term forecasting will be unreliable since the environment is in reality behaving chaotically. For business users and their organizations this means that the ability to act swiftly based upon changes in the environment is the key determining factor for success and failure.

The framework proposed as part of the Computer Aided Leadership & Management (CALM) philosophy [36] is specifically made to cope with the challenges faced by leaders and managers since they effectively operate in a world of chaos due to the globalization of commerce and connectivity; one could say that in this chaotic world, the ability to continuously act is far more determinative for success than the ability to long-term forecast. The idea in CALM is to take the Observation-Orientation-Decision-Action (OODA) loop (originally pioneered by "Top Gun" fighter pilot John Boyd in the 1950s), and integrate business intelligence (BI) technologies to drastically increase the speed with which a user in an organization cycles through the OODA loop.

Using the framework proposed in the CALM philosophy, any organization and its purpose can be described as a number of OODA loops (Figure 1.1) that are continuously cycled by users and conform to one or more Key Performance Indicators (KPI's) that define their success. The rationale is that if these four phases are cycled continuously and as fast as possible for every business user in an organization, then

the organization is optimizing its resources and maximizing its ability to deal with both problems and opportunities. If we succeed in allowing a user in an organization to move as fast as possible all the way from the observation phase to the action phase, we are basically using information technology to channel the core competency of that organization as effectively into its environment as possible. In CALM we use the following definitions of the individual phases:

Observation in this phase we use technology to look at the data with an expectation of what it should be.

Orientation in this phase we look at the data in different ways depending on what it shows, typically this phase is initiated after something in the observation phase has proven to be different from what we expected.

Decision this phase is currently undertaken primarily by human intelligence, but the results found in the Data Warehouse can be evaluated against other external or internal -typically unstructured- data.

Action based on the decision made, we seek to implement the course of action chosen.

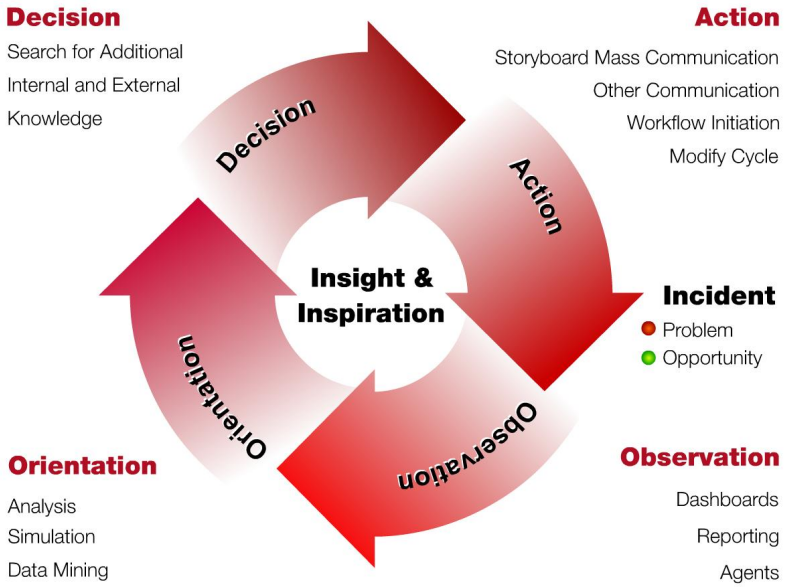


Figure 1.1: The OODA loop aligned with the different BI disciplines.

Having established a number of OODA loops with a number of KPI's assigned, we should ideally provide the users with technology that can help them identify patterns that can give early warnings whenever a KPI is threatened, and thus allow provide the user with a time advantage in the observation phase in order to take corrective action before our KPI is impacted. In the CALM philosophy, we refer to such a technology as a *sentinel*.

One could say that sentinels, once identified, reduce the time from observation to action, and that the discovery of sentinels reduces the time in the orientation phase. One way to look at the sentinel concept is that it expands the "horizon" by allowing the user to see data from the external environment, and not only for the internal performance of the organization. The idea behind placing sentinels at the outskirts of the data available for an organization seeks to harness both these ways of improving reaction time and thus organizational competitiveness. Metaphorically, what we seek to do for navigation of organizations with sentinels, is what radars do for navigation of ships.

This thesis is based on research in pursuit of making the sentinels described in the CALM philosophy into a reality. The goal of this research was to develop the sentinel technology and to make it available for the global 274,000 users of the TARGIT Business Intelligence Suite (TARGIT BI Suite). The TARGIT BI Suite is a commercial software package for business intelligence, which is specifically directed at organizations that seek to improve the speed in the OODA loop for its employees. The primary driver for the TARGIT BI Suite has been a strong focus on usability as well as on the integration of the traditional business intelligence disciplines, i.e., dashboarding, analysis, and reporting. Therefore, the sentinel research is concerned with limiting the complexity for the end users as much as possible, as well as ensuring integration with the industrial standard multidimensional database environment, in addition to transferring sentinels from concept to reality. In this context, it should be noted that bringing data mining capabilities to end-users has been a dream for the software industry since the late 1990s, yet no company had succeeded in delivering to this promise when this project was initiated (see Chapter 5).

1.2 Contributions

Chapter 2 defines the concept of *sentinel rules* for multi-dimensional data for the first time. The sentinel rules in this chapter are later referred to as *regular sentinel rules* or *simple sentinel rules* since they can only describe a relationship between two measures. For instance, a surge in negative blogging about a company could trigger a sentinel rule warning that revenue will decrease within two months, so a new course of action can be taken. We demonstrate, by example, how sentinel rules work at the

schema level as opposed to the *data level*, and their operations on data *changes* as opposed to absolute data values. In this chapter, the first sentinel mining algorithm is presented along with its implementation in SQL. Although the implementation of sentinel mining is straight forward, compared to what will be presented in the following chapters, we demonstrate that this particular implementation scales linearly on large data volumes. Another important contribution in this chapter is the demonstration of the distinct differences between sentinel mining and sequential pattern mining as well as its difference from correlation techniques. In this case it is specifically demonstrated that sentinel mining can find strong and useful sentinel rules that would otherwise be hidden when using sequential pattern mining or correlation techniques.

Chapter 3 extends the formal definitions from previous chapter into *generalized sentinel rules* which we also refer to as just *sentinels*. With the generalization, sentinels are now capable of describing relationships with multiple measures in a multi-dimensional data cube. Specifically, generalized sentinel rules represent *schema level* relationships where *changes over time* in multiple so-called *source measures* are followed by similar changes over time in a so-called *target measure*. Compared to the previous example, these sentinels could notify a user that revenue might drop within two months if an increase in customer problems combined with a decrease in website traffic is observed. If the vice versa also holds, i.e., a decrease in customer problems combined with an increase in website traffic is expected to lead to an increase in revenue, we have a *bi-directional* sentinel, which has a higher chance of being causal rather than coincidental (The desirability of bi-directionality is further explained in Chapters 4 and 5). In this chapter a novel quality measure, *Balance*, which describes the degree of bi-directionality in a sentinel is introduced. Furthermore, the quality measure, *Score*, which is used to assess the overall quality of a sentinel, is also introduced. Using greedy algorithms that seek for the highest *Score*, we are able to extend the abilities of sentinel mining to combine multiple measures into better sentinels as well as auto-fitting the best warning period. For optimization, we introduce a novel *Reduced Pattern Growth* (RPG) optimization, which in combination with a hill climbing optimization, contributes to a significantly more efficient sentinel mining algorithm, *SentHiRPG*. The *SentHiRPG* algorithm is demonstrated to be efficient and to scale to very large datasets on both real and synthetic data.

Chapter 4 is a user side demonstration of sentinel mining as it has been implemented in the TARGIT BI Suite. In addition, this chapter gives a more intuitive presentation of the sentinel mining concept than the previous chapters. The focus of this chapter is to demonstrate how sentinel mining can be conducted by casual users without any prior technical knowledge. The built-in context awareness of the TARGIT BI Suite, which is a facilitator for a fast OODA cycle, is demonstrated to facilitate an intuitive way for users to conduct sentinel mining. Specifically, is is demonstrated

how a user with a minimum of input can mine sentinels and subsequently schedule them for future notifications.

Chapter 5 describes the underlying technology that is presented in Chapter 4, and it describes the underlying implementation that has been done in the TARGIT BI Suite version 2K10 in order to provide the users with sentinel mining. This chapter extends the intuitive presentation from the previous chapter and describes in more detail how sentinels are mined from both a data and a user perspective. This chapter also describes the architecture used in the implementation of the, *SentHiRPG*, sentinel mining algorithm in the TARGIT BI Suite. Specifically, the chapter describes in detail the TARGIT ANTserver component layer by layer in which the sentinel mining algorithm has been implemented. Following the implementation, feedback following the version 2K9 launch from both customers and market analysts is described. The industrial implementation of sentinel mining is tested on a real-world operational data warehouse located in TARGIT A/S. On this data warehouse, it is demonstrated through extensive experiments, that mining and usage of sentinels is feasible with good performance for the typical users on a real, operational data warehouse.

Chapter 6 introduces a novel and very efficient algorithm, *SentBit*, that is 2–3 orders of magnitude faster than the *SentHiRPG* algorithm. The general idea is to encode all measure changes, referred to as *indication streams*, into bitmaps, and subsequently to conduct the entire mining process as bitmap operations. The bitmap operations include so-called *indication joins*, that rapidly combine measure changes using AND operations. By representing the entire sentinel mining problem by bitmap operations, we are able to effectively utilize modern CPU specific instructions dedicated to counting the number of bits set in a given stream of indications. This means that all quality measures needed to evaluate a given sentinel can be rapidly calculated. In addition, the multi-core architectures available on modern processors allow us to run the joins and evaluations of multiple sentinels in parallel. The *SentBit* algorithm optimized by CPU specific instructions and the parallelization is evaluated to gain unsurpassed efficiency by a factor of 2–3 orders of magnitude over the *SentHiRPG* algorithm. We demonstrate that the *SentBit* algorithm scales efficiently to very large datasets, which is verified by extensive experiments on both real and synthetic data.

Chapter 7 presents a highly efficient bitmap-based algorithm, *SentBMD* for discovery of so-called *multidimensional sentinels*. Compared to earlier examples, multidimensional sentinels notify users based on previous observations in subsets of a multidimensional data cube, e.g., that revenue might drop within two months if an increase in customer complaints in USA (drilldown into geography dimension) combined with a decrease in the money invested in customer support for laptop computers (drilldown into product dimension) is observed. In other words, compared to prior algorithms, the *SentBMD* algorithm is able to discover patterns that include the hierarchical dimensions in an OLAP cube. Compared to *SentBit* in previous chapter,

the *SentBMD* algorithm combine the schema and the data levels, meaning that each measure change in the rule can hold for either *subsets* or the entire cube. In addition, the *SentBMD* algorithm allows source measures to be progressively input during the mining process which means that the encoding of source measure changes can run in parallel with the mining process. *SentBMD* is thus even more efficient in the utilization of multiple cores on modern CPUs. Finally, this chapter demonstrates through extensive experiments that the *SentBMD* algorithm is significantly more efficient than *SentBit* when running on comparable "flat datasets", and that *SentBMD* scales efficiently to very large real and synthetic datasets.

Appendix A explains in more detail how sentinels are linked to the CALM philosophy, and it uses the Observation-Orientation-Decision-Action (OODA) concept as a mean to identify three new desired technologies in business intelligence applications (whereof sentinels is one) that improve the speed and quality in the decision making processes.

A final contribution of this work is that bitmapped sentinel mining has been made commercially available in the TARGIT BI Suite version 2K10 SR1. A fully functional demo version can be downloaded from `www.targit.com`.

The thesis is organized as a collection of individual papers. The chapters are organized such that material that is used in or motivates the work in another chapter appears first. However, each chapter/appendix is self-contained and can be read in isolation. The chapters have been slightly modified during the integration such that, for example, their bibliographies have been combined to one and references to "this paper" have been changed to references to "this chapter". There are some overlaps between the introductional parts of the chapters. Specifically, the motivation part of the introduction in Chapters 2–7 is overlapping. In addition, the description of the sentinel concept is overlapping in Chapters 4 and 6. Finally, given the integration of the papers as progressing research, the related work mentioned in each chapter overlaps with the related work of chapter based on the previous paper. When reading the thesis cover to cover, the following sections can be skipped: Section 4.3, "The SentHiRPG Algorithm" paragraph in Section 5.3, and Section 6.2.

The papers included in the thesis are listed below. Chapter 2 is based on Paper 1, Chapter 3 is based on Paper 2 and so on until Chapter 7 which is based on Paper 6. Appendix A is a short paper from an invited talk at the 10th ACM international workshop on Data warehousing and OLAP in 2007 [37].

1. M. Middelfart and T.B. Pedersen. Discovering Sentinel Rules for Business Intelligence. DB Tech Report no. 24, `dbtr.cs.aau.dk` [38]. Short version in *Proceedings of the 20th International Conference on Database and Expert Systems Applications (DEXA)*, pp. 592–602, 2009 [39].

2. M. Middelfart, T.B. Pedersen, and J. Krogsgaard. Efficient Discovery of Generalized Sentinel Rules. In *Proceedings of the 21st International Conference on Database and Expert Systems Applications (DEXA)*, Part II, pp. 32–48, 2010 [40].

3. M. Middelfart and T.B. Pedersen. Using Sentinel Technology in the TARGIT BI Suite. In *Proceedings of the VLDB Endowment 3(2)*: 1629–1632, 2010 [41].

4. M. Middelfart and T.B. Pedersen. Implementing Sentinel Technology in the TARGIT BI Suite. *In submission* [42].

5. M. Middelfart, T.B. Pedersen, and J. Krogsgaard. Efficient Sentinel Mining Using Bitmaps on Modern Processors. *In submission* [43].

6. M. Middelfart, T.B. Pedersen, and J. Krogsgaard. Multidimensional Sentinel Mining Using Bitmaps. *In submission* [44].

1.3 Related Work

In general, there has been very little theory about how best to apply computing to the decision process in a data warehousing context. Kimball's "Analytical Cycle" [30] is the exception, however compared to the CALM [36] theory it does not strive for human/computer synergies. In addition, some work has cited John Boyd of the OODA loop [32], however none have used the OODA loop as the primary driver for the implementation and integration of BI disciplines in a decision process such as it was done with CALM. The ability to predict a certain incident based on a number of previous incidents have been well explored in *association rule mining* [2], also known as *frequent pattern mining*. Perhaps the most popular real-world application of association rules is the traditional basket analysis case, where the frequency of items purchased together in a supermarket is analyzed and the rules generated are applied when physically positioning the products inside the store. Today, we also see these types of rules applied in on-line stores that recommend additional products when buying one based on the rationale: "other people who bought this product did also buy this additional product". *Sequential pattern mining* has a lot of similarity to association rule mining, however, it allows a time period to pass between the "baskets". It should of course be noted that "basket" at this stage is simply a label for a group of incidents happening together. In general, one can say that these mining techniques are concerned with identifying an incident, a premise, that leads to another incident, the consequent. In other words, these techniques are focused on co-occurrence patterns within absolute data values for categorical data, and by using these patterns to create a number of rules to be used for prediction of behavior within a given domain.

With regards to association rule mining, significant effort has been put into the optimization of the original Apriori algorithm [3]. These improvements have contributed with more efficiency and lower memory usage by growing a compressed frequent pattern tree in memory [25]. The addition of a period between the incidents in sequential pattern mining adds to the complexity of association rules, meaning that an Apriori approach is even more costly [5], and thus new approaches to improving the performance of mining sequential patterns have emerged [24, 48, 49, 57]. In pattern mining it has proven a challenge that a huge number of more or less interesting rules are output, and thus improvements to the selection process of thresholds that determine the interestingness of rules has been introduced to association rule mining [58]. Similarly, sequential pattern mining process has also seen improvements in terms of applying constraints [21], as well as a querying-type approach [20] that makes the mining process more interactive. Another approach to limiting the output of pattern mining to only the most relevant rules have been explored in the mining of non-derivable frequent itemsets [12]. This approach seek to limit the number of association rules (frequent patterns) in the output, and thus allow the user of the algorithm to only focus on the most important of association rules, e.g., those rules that are unique and from which all other rules can be derived. Additionally, similar compression of the output has been explored for sequential patterns with so-called conjunctive sequential patterns [53] that seek to condense the output of sequential pattern mining similarly to the mining of non-derivable frequent itemsets mentioned above, e.g., the two sequential patterns $\{(a)(b)(a,c),(a,b)(c)\}$ can both be described by the conjunction $(b)(c)$ that supports both patterns. Another aspect of sequential pattern mining has been the various approaches to handling the period of the sequence, e.g., involving multiple time granularities [9] or allowing for partial periodicity of patterns [23].

Gradual rule mining [7,10,27,31] is a process much like association rules, where the categorical data are created by mapping numerical data to fuzzy partitions, with the primary objective of describing the absolute values of a measure, and thereby identify complete *overall* relationships.

Correlation [26,60] and *regression* [4] techniques are other approaches to prediction, these techniques identify trends that can be used to create a mathematical model of the domain. The time dimension is typically one of the input parameters in the model, and by extending the time dimension into the future, the model can then be used to output predicted outcomes.

The introduction of novel CPU architectures and multi-dimensional databases have extended all of the techniques above. In [6], association rule mining exploits a parallel shared-nothing multiprocessor system. In addition, multi-core parallelism has been applied to gradual rule mining [31]. In general, the use of parallelization has been applied successfully in data warehousing in order to deal with the huge vol-

umes of data that resides in these systems [17,18]. With regards to multi-dimensional databases, these data structures have been explored in *multi-dimensional pattern mining* [51,52] and *multi-level frequent pattern mining* [19] which apply the same techniques to more dimensions and dimensional levels than just one. Regression has also been used in multi-dimensional databases, e.g., Bellwether Analysis can be used on historical sales data to identify a leading indicator that can predict the global sales of a new product, for which only a short period of data is available [13,14].

Sentinel mining is different from association rule and sequential pattern mining since its primary focus is to identify strong relationships in smaller *subsets* of the data in order to warn users about potential problems that need attention and *action*. The fact that sentinels represent a subset of the data rather than a global trend means that a sentinel is able to stimulate a specific action, since the user is presented with the specific causal relationship leading to the threat. One could say that sentinels are concerned with mining relationships between *change points* in the data, and based on these we seek to predict and warn about potential turning points (desired or undesired) for critical measures. The motivation for bellwether analysis is similar to that of sentinel mining, since the idea is to identify early indicators of future outcome for a given measure. However, similar to gradual rules, these techniques are also concerned with the absolute values of a measure, as opposed to sentinels that are based on changes in the measure values. With regards to the output, sentinels are more *specific "micro-predictions"*, i.e., strong rules that hold for a subset of the data and stimulate specific actions, and are thus complementary to these techniques. Sentinels are therefore useful for detecting incidents and generating warnings whenever changes (that would otherwise go unnoticed) in a relevant source measure occur. The differences described for bellwether analysis also hold for correlation and regression techniques in general, and in Chapter 2, we provide a concrete example to demonstrate the difference between these techniques and sentinel mining.

The main difference between association rule [2], sequential pattern mining [5], and multi-level frequent pattern mining [52] is that sentinel mining is working on *relative changes in data* as opposed to *absolute data values*. In a multi-dimensional setup, association rule mining typically works on *categorical data*, i.e., dimension values, whereas sentinels work on *numerical data* such as measure values. An important property of sentinels is the ability to describe *bi-directional* relationships, which essentially means that sentinels are able to predict in both directions on the measure targeted for prediction. Association rules, sequential patterns, and gradual rules do not have this property. In addition, the schema-level nature of sentinels, meaning that they describe relationships between measures rather than data values, is another differentiator. The combination of schema-level and bi-directional rules based on relative changes in data allows us to generate fewer, more general, rules. If we assume that we could mine a "sentinel-like" rule using sequential pattern mining, such a rule

would only be *uni-directional*, as opposed to sentinels that can be *bi-directional* and thus *stronger* with a greater chance of being causal rather than coincidental as elaborated in Chapter 5. In Chapter 2 we specifically provide a concrete, realistic example where nothing useful is found using these techniques, while sentinel mining *do* find meaningful rules.

With regards to the optimizations applied in sentinel mining, the schema level nature of sentinels, including the focus on data changes, gives rise to the various optimizations that are applied in Chapters 3, 6, and 7. In Chapter 3, the table of combinations (TC) and the reduced pattern growth (RPG) optimization, and such optimizations can therefore not be offered by sequential pattern mining or other known optimizations for simpler "market basket"-type data such as [11]. Similarly, the ability to combine source measures into better sentinel rules, adds to the distance between sentinels and optimizations offered in prior art such as [3, 24, 46, 48, 49, 57, 59]. In Chapters 6 and 7, the bitmapped encoding of measures and the processor specific optimizations also rely on the schema level property of sentinels.

Aside from the scientific community, sentinels are also distinctively different from the traditional algorithms offered in the business intelligence industry. The industry analyst Gartner, is a well-known authority in the business intelligence industry, and the Magic Quadrant analysis is by many considered the most influential description of the top international vendors. The Magic Quadrant analysis for 2010 [54] categorizes 15 vendors as either "market leaders" (7 companies), "challengers" (3 companies), or "niche players" (5 companies). The companies identified as "market leaders" are: IBM, Oracle, Microsoft, SAS, SAP, Information Builders, and MicroStrategy. Gartner categorizes features such as sentinels as "predictive modeling and data mining", and all "market leaders" have features within this category. However, only SAS seems to be significantly differentiated from the other "market leaders" in this category with a more comprehensive predictive offering since the company originated from forecasting and predictive modeling, whereas the other companies started as DBMS providers or as providers of reporting centric solutions. Out of all features offered by the "market leaders" [8,28,34,35,47,56], the algorithms that to some extent resemble the functionality of sentinels are *association rules*, *sequential patterns*, and *regression techniques*. This also holds for the "challenger" Tibco, which is the only other noteworthy company with "predictive modeling and data mining" features. As explained above, these competing techniques are distinctly different from the sentinel technology implemented by TARGIT. Moreover, TARGIT is a new entrant in the "niche player" category of the Magic Quadrant, and with reference to Gartner's statement below, the sentinels of TARGIT are also perceived as a unique feature in the Magic Quadrant analysis [54].

"The introduction of an innovative alerting solution, called Sentinels (essentially prediction-based rules), enables an end user to react quickly to alerts for certain indicators. Through the combination with Targit's desktop alerts, a user gets an early-warning notification when a predefined rule has been violated and the user can proactively take corrective measures. This capability adds to Targit's attractiveness for end users."

In general, TARGIT is seen by Gartner as a "niche player" with a strong foothold in the mid-market, and with a unique position in its approach to BI usability. This is very much in line with the fast OODA loops facilitated by BI mentioned in the previous section. Therefore the entire concept of sentinels is unique from a market perspective.

Chapter 2

Discovering Sentinel Rules for Business Intelligence

This chapter proposes the concept of *sentinel rules* for multi-dimensional data that warns users when measure data concerning the external environment changes. For instance, a surge in negative blogging about a company could trigger a sentinel rule warning that revenue will decrease within two months, so a new course of action can be taken. Hereby, we expand the window of opportunity for organizations and facilitate successful navigation even though the world behaves chaotically. Since sentinel rules are at the schema level as opposed to the data level, and operate on data *changes* as opposed to absolute data values, we are able to discover strong and useful sentinel rules that would otherwise be hidden when using sequential pattern mining or correlation techniques. We present a method for sentinel rule discovery and an implementation of this method that scales linearly on large data volumes.

2.1 Introduction

The Computer Aided Leadership and Management (CALM) concept copes with the challenges facing managers that operate in a world of chaos due to the globalization of commerce and connectivity [36]; in this chaotic world, the ability to continuously act is far more crucial for success than the ability to long-term forecast. The idea in CALM is to take the Observation-Orientation-Decision-Action (OODA) loop (originally pioneered by "Top Gun"[1] fighter pilot John Boyd in the 1950s [32]), and in-

[1]Colonel John Boyd was fighter instructor at Nellis Air Force Base in Nevada, the predecessor of U.S. Navy Fighter Weapons School nicknamed "Top Gun".

tegrate business intelligence (BI) technologies to drastically increase the speed with which a user in an organization cycles through the OODA loop. Using CALM, any organization can be described as a set of OODA loops that are continuously cycled to fulfill one or more Key Performance Indicators (KPI's). One way to improve the speed from observation to action is to expand the "horizon" by allowing the user to see data from the external environment, and not only for the internal performance of the organization. Another way is to give early warnings when factors change that might influence the user's KPI's, e.g., revenue. Placing "sentinels" at the outskirts of the data available seeks to harness both ways of improving reaction time and thus organizational competitiveness.

A sentinel rule is a relationship between two measures, A and B, in an OLAP database where we know, that a change in measure A at one point in time affects measure B within a certain *warning period*, with a certain confidence. If such a relationship exists, we call measure A the *source measure*, and measure B the *target measure*. Usually, the target measure is, or contributes to, a KPI. The source measure ideally represents the external environment, or is as close to the external environment as possible. Examples of source measures for an organization could be: the number of negative blog entries (external), the number of positive articles in papers (external), or the number of complaints from customers (internal, yet as close to external as possible). Examples of target measures could be: revenue or contribution margin. Imagine a company selling a product globally where we discovered the sentinel rule: "IF negative blogs go up THEN revenue goes down within two months AND IF negative blogs go down THEN revenue goes up within two months". Assume that Google is searched daily for negative blogs, and the number of negative blogs is stored in the company's OLAP database. Also, the company's BI solution can notify users based on sentinel rules. When a user receives notification that the number of negative blogs goes up, he will know that revenue will go down in two months with a certain confidence. Depending on the situation, the user might have a number of evasive actions such as: post positive blogs to sway the mood, or reduce cost to cope with a reduction in revenue. Regardless of the action, the sentinel rule has raised awareness of a problem and reduced the time from observation to action. Metaphorically, sentinels seek to do for organizations what radars do for navigation of ships.

The novel contributions in this chapter include the sentinel rule concept, and an algorithm that discover sentinel rules on multi-dimensional data. We give a formal definition of sentinel rules, and we define the indication concept for rules and for source and target measures. In this context, we provide a contradiction elimination process that allows us to generate more general rules that are easy to interpret. We also provide a useful notation for sentinel rules. We conduct several experiments to validate that our algorithm scales linearly on large volumes of synthetic and real-world data and on databases with high complexity in terms of the number of mea-

sures. Since sentinel rules operate on data changes as opposed to absolute data values, and are at the *schema level* as opposed to the *data level* (such as association rules and sequential patterns), we can find strong rules that neither association rules nor sequential pattern mining would find. In addition, we found sentinel rules to be complementary to correlation techniques, since our solution finds "micro-predictions" that hold for a smaller subset within a dataset; using correlation techniques alone such rules would be "hidden in the average". We believe that we are the first to propose the concept of sentinel rules, and to provide an algorithm and implementation for discovering them.

The next section presents the formal definition, Section 2.3 presents an algorithm including an assessment of its complexity. Section 2.5 presents a scalability and a qualitative study. Section 2.6 presents the work related to the discovery of sentinel rules.

2.2 Problem Definition

Running Example: Imagine a company that sells products world-wide, and that we, in addition to the traditional financial figures such as revenue, *Rev*, have been monitoring the environment outside our organization and collected that information in three measures. The measure *NBlgs* represents the number of times an entry is written on a blog where a user is venting a negative opinion about our company or products. The measure *CstPrb* represents the number of times a customer contacts our company with a problem related to our products. The measure *WHts* represents the number of hits on our website, and this figure has been cleansed in order to represent human contact exclusively, eliminating traffic by robots etc.

T: Time	D_2: Region	M_1: NBlgs	M_2: CstPrb	M_3: WHts	M_4: Rev
2007-Q1	Asia	20	50	1,000	10,000
2007-Q2	Asia	21	45	1,500	9,000
2007-Q3	Asia	17	33	2,000	11,000
2007-Q4	Asia	15	34	2,500	13,000
2007-Q1	EU	30	41	3,000	20,000
2007-Q2	EU	25	36	3,500	25,000
2007-Q3	EU	22	46	4,000	28,000
2007-Q4	EU	19	37	4,500	35,000
2007-Q1	USA	29	60	5,000	50,000
2007-Q2	USA	35	70	5,500	55,000
2007-Q3	USA	40	72	6,500	45,000
2007-Q4	USA	39	73	7,500	40,000

Table 2.1: Example dataset.

In Table 2.1 we see a subset from our database, representing each quarter in year 2007 across three geographical regions. It should be noted that a subset like Table 2.1 can easily be extracted from a multi-dimensional database, i.e., if the desired data are the base level of the database no processing is needed, if the desired levels are higher than the base level, the data might or might not be preaggregated. However, both extraction and aggregation are typically basic built in functions of any multi-dimensional database. The three measures: NBlgs, CstPrb and WHts, representing the external environment around our company, have been presented along with the internal measure, Rev, representing our Revenue. The variable names: $T, D_2, M_1...M_4$ have been assigned to the dimensions and measures in order to create transparency to the formal definition in Section 2.2.

We are interested in discovering whether we can use any of the external measures to predict a future impact on the internal Revenue measure; in other words we are looking for sentinel rules where one of the measures $M_1...M_3$ can give us an early warning about changes to M_4. To distinguish between which measures are "causing" the other, we call the measures $M_1...M_3$ *source measures* and the measure M_4 is called the *target measure*.

Formal Definition: Let C be a multi-dimensional data cube containing a set of dimensions: $D = \{D_1, D_2...D_n\}$ and a set of measures: $M = \{M_1, M_2...M_p\}$. We denote the members of the dimensions in D by $d_1, d_2...d_n$ and we denote the corresponding *measure values* for any combination of dimension members by $m_1, m_2...m_p$. A measure value is a function, M_i, that returns the value of a given measure corresponding to the dimension members it is presented with. We will now provide a series of definitions that define a source measure, A, is a *sentinel* for a target measure, B, i.e., a guarded watchtower from which we monitor A in order to know about changes ahead of time to B. The sentinel rule between A and B is denoted $A \rightsquigarrow B$. We assume, without loss of generality, that there is only one time dimension, T, in C, and that $T = D_1$, and subsequently $t = d_1$. A fact, f, in C is then defined as:

$$f = (t, d_2, d_3...d_n, m_1, m_2...m_p) \tag{2.1}$$

Given a fact f, the measure M_i is a function $M_i(t, d_2, d_3...d_n) = m_i$. The "dimension" part of f, $(t, d_2, d_3...d_n)$, is called a cell. The *shifting* of a fact f, f', is a fact with the same non-time dimension values $(d_2...d_n)$ as f, but for time period $t + o$, if it exists in C, i.e., a period of o members later on the time dimension. We denote the *offset*, o, and define the function as:

$$\mathit{Shift}(C, f, o) = f' = (t + o, d_2, d_3...d_n, m'_1, m'_2...m'_p) \text{ if } f' \in C \tag{2.2}$$

Since we are interested in the change in data, we introduce the *measure difference function*, *Diff*. With *Diff*, we find the relative changes to each of the measures during the time period specified by the offset. *Diff* is defined as follows:

$$\begin{aligned} \mathit{Diff}\,(C, f, o) = \; & (t, d_2, d_3 ... d_n, \frac{m'_1 - m_1}{m_1}, \frac{m'_2 - m_2}{m_2} ... \frac{m'_p - m_p}{m_p}) \\ & \text{where } f = (t, d_2, d_3 ... d_n, m_1, m_2 ... m_p) \; \wedge \; f \in C \; \wedge \\ & f' = \mathit{Shift}(C, f, o) = (t + o, d_2, d_3 ... d_n, m'_1, m'_2 ... m'_p) \wedge f' \in C \end{aligned} \tag{2.3}$$

Given a threshold, α, we say that $x \in \mathit{Diff}\,(C, f, o)$ is an *indication* on a measure, M_i, if:

$$x = (t, d_2, d_3 ... d_n, \frac{m'_1 - m_1}{m_1}, ..., \frac{m'_i - m_i}{m_i}, ..., \frac{m'_p - m_p}{m_p}) \wedge |\frac{m'_i - m_i}{m_i}| \geqq \alpha \tag{2.4}$$

We say that an indication on M_i, x, is *positive*, denoted M_i▲, when $\frac{m'_i - m_i}{m_i} > 0$ and consequently that an indication, x, is *negative*, denoted M_i▼, when $\frac{m'_i - m_i}{m_i} < 0$. We define a wildcard, $*$, meaning that M_i* can be either M_i▲ or M_i▼.

In our running example, when assessing whether a relationship exists, we are not concerned with minor fluctuations, so we define a threshold of 10%, meaning that a measure has to change at least 10% up or down in order to be of interest. Furthermore, given the dataset we have, we are interested in seeing the changes that occur over *quarters* as presented in Table 2.1. This means that we set the threshold $\alpha = 10\%$ and then the offset *o = 1 Quarter*. In Table 2.2, we have calculated the changes from each quarter to the next and subjected each change to an evaluation against the threshold of 10% change. We denote positive indications by ▲ and subsequently negative by ▼, if a change is less than 10% in either direction it is deemed "neutral". Please note that since we are dealing with *changes* between periods, we naturally get one less row for each region.

T: Time	D_2: Region	M_1: NBlgs	M_2: CstPrb	M_3: WHts	M_4: Rev
'07:Q1→Q2	Asia	neutral	M_2▼	M_3▲	M_4▼
'07:Q2→Q3	Asia	M_1▼	M_2▼	M_3▲	M_4▲
'07:Q3→Q4	Asia	M_1▼	neutral	M_3▲	M_4▲
'07:Q1→Q2	EU	M_1▼	M_2▼	M_3▲	M_4▲
'07:Q2→Q3	EU	M_1▼	M_2▲	M_3▲	M_4▲
'07:Q3→Q4	EU	M_1▼	M_2▼	M_3▲	M_4▲
'07:Q1→Q2	USA	M_1▲	M_2▲	M_3▲	M_4▲
'07:Q2→Q3	USA	M_1▲	neutral	M_3▲	M_4▼
'07:Q3→Q4	USA	neutral	neutral	M_3▲	M_4▼

Table 2.2: Indications between quarters.

$$ST(C, o, w) = \{(\mathit{Diff}(C, f, o), \mathit{Diff}(C, \mathit{Shift}(C, f, w), o)) | f \in C\} \quad (2.5)$$

A Source-Target Set, ST, is defined as paired indications of changes over time, where the source and target measures have been shifted with the offset, o. The target measures have additionally been shifted with a *warning period*, w, which is the time-frame after which we should expect a change on a target measure, after an indication on a source measure has occurred. We say that $(x, x') \in ST(C, o, w)$ *supports* the *indication rule* $A\blacktriangle \rightarrow B\blacktriangle$ if x is an indication of $A\blacktriangle$ and x' is an indication of $B\blacktriangle$. In this case, we also say that *x* supports $A\blacktriangle$ and x' supports $B\blacktriangle$. The *support* of an indication rule is the number of $(x, x') \in ST(C, o, w)$ which supports the rule. The support of indication rules $A\blacktriangledown \rightarrow B\blacktriangledown$, $A\blacktriangle \rightarrow B\blacktriangledown$ and $A\blacktriangledown \rightarrow B\blacktriangle$ as well as the support for indications $A\blacktriangledown$ and $B\blacktriangledown$ are defined similarly. We denote the support of an indication and an indication rule by ***IndSupp*** followed by the name of the indication or indication rule, respectively, e.g., $\mathit{IndSupp}_{A\blacktriangle}$ and $\mathit{IndSupp}_{A\blacktriangle \rightarrow B\blacktriangle}$.

A sentinel rule is an *unambiguous* relationship between A and B, thus we must first eliminate contradicting indication rules, if such exist, before we have a sentinel rule. We refer to this process as the *contradiction elimination process*, and we use it to remove indication rules with the same premise, but a different consequent, and vice versa, e.g., if both $A\blacktriangle \rightarrow B\blacktriangle$ and $A\blacktriangle \rightarrow B\blacktriangledown$ or if both $A\blacktriangle \rightarrow B\blacktriangle$ and $A\blacktriangledown \rightarrow B\blacktriangle$ are supported. To eliminate such contradictions, we pair the indication rules in two sets that do not contradict each other, and we denote these sets by $A \rightarrow B$ and $A \rightarrow \mathit{inv}(B)$, as follows: $A \rightarrow B = \{A\blacktriangle \rightarrow B\blacktriangle, A\blacktriangledown \rightarrow B\blacktriangledown\}$ and $A \rightarrow \mathit{inv}(B) = \{A\blacktriangle \rightarrow B\blacktriangledown, A\blacktriangledown \rightarrow B\blacktriangle\}$. Here *inv* indicates an inverted relationship between the indications on A and B, e.g. if $A\blacktriangle$ then $B\blacktriangledown$, and vice versa.

For the purpose of being able to deduct the support of the indication rule(s) we eliminate, we define functions for returning the premise and the consequent indication, ***IndPrem*** and ***IndCons***, from an indication rule $A\blacktriangle \rightarrow B\blacktriangle$ as follows: $\mathit{IndPrem}(A\blacktriangle \rightarrow B\blacktriangle) = A\blacktriangle$ and $\mathit{IndCons}(A\blacktriangle \rightarrow B\blacktriangle) = B\blacktriangle$. Furthermore, we define the complement of an indication as follows: $\overline{A\blacktriangle} = A\blacktriangledown$ and $\overline{A\blacktriangledown} = A\blacktriangle$. We can now define a contradicting indication rule as a function, ***ContraRule***, for an indication rule, ***IndRule***, as follows:

$$\mathit{ContraRule}(\mathit{IndRule}) = \mathit{IndPrem}(\mathit{IndRule}) \rightarrow \overline{\mathit{IndCons}(\mathit{IndRule})} \quad (2.6)$$

$$\mathit{ElimSupp}(\mathit{IndRule}) = \mathit{IndSupp}_{\mathit{IndRule}} - \mathit{IndSupp}_{\mathit{ContraRule}(\mathit{IndRule})} \quad (2.7)$$

The support after elimination, ***ElimSupp***, of an indication rule, ***IndRule***, where the support of the contradicting indication rule, ***ContraRule(IndRule)***, has been eliminated can be calculated as shown in Formula (2.7).

$$MaxRule = \begin{cases} \{IndRule_i \mid IndRule_i \in A \rightarrow B \ \wedge \ ElimSupp(IndRule_i) > 0\} \\ \qquad\qquad \text{if } IndSupp_{A \rightarrow B} >= IndSupp_{A \rightarrow inv(B)}, \\ \{IndRule_i \mid IndRule_i \in A \rightarrow inv(B) \ \wedge \ ElimSupp(IndRule_i) > 0\} \\ \qquad\qquad \text{if } IndSupp_{A \rightarrow B} < IndSupp_{A \rightarrow inv(B)}. \end{cases} \tag{2.8}$$

MaxRule is the set of indication rule(s), $IndRule_i$, in the set ($A \rightarrow B$ or $A \rightarrow inv(B)$) with the highest ***IndSupp*** and where $ElimSupp(IndRule_i) > 0$. With ***MaxRule***, we have identified the best indication rule(s) for a sentinel rule that represents an unambiguous relationship between A and B, i.e., the non-contradicting indication rules with the highest ***ElimSupp***. In other words, we have eliminated the *contradicting* indication rules where the premise contradicts the consequent, as well as the *orthogonal* indication rules where different premises have the same consequent. If the ***MaxRule*** set consists of only one indication rule, we refer to the sentinel rule based on this as a ***uni-directional*** rule.

We denote the support of a sentinel rule by ***SentSupp***, followed by the name of the sentinel rule, e.g., $SentSupp_{A \rightsquigarrow B}$. For a potential sentinel rule, $A \rightsquigarrow B$, we define ***SentSupp*** as the sum of the support of source measure indications for the indication rule(s) contained in the sentinel rule:

$$SentSupp_{A \rightsquigarrow B} = \begin{cases} IndSupp_{A\blacktriangle} & \text{if } A\blacktriangledown \rightarrow B* \notin MaxRule, \\ IndSupp_{A\blacktriangledown} & \text{if } A\blacktriangle \rightarrow B* \notin MaxRule, \\ IndSupp_{A\blacktriangle} + IndSupp_{A\blacktriangledown} & \text{otherwise.} \end{cases} \tag{2.9}$$

In Formula (2.9) we note the difference between the support of an indication rule, ***IndSupp***, and a sentinel rule, ***SentSupp***. Specifically, when calculating the support of a sentinel rule, $SentSupp_{A \rightsquigarrow B}$, we only consider the support of indications on the source measure (the premise), $A\blacktriangle$ and $A\blacktriangledown$. With indication rules, both indications on the source and target measure needs to occur. The reason is, that the consequential support of indications on the target measure, $B\blacktriangle$ or $B\blacktriangledown$, is taken into consideration when calculating the confidence of the sentinel rule in Formula (2.10). In the case of a uni-directional rule (the two first cases) we only consider the support of indications on the source measure that have the same direction as the one indication rule in ***MaxRule***; this is done in order not to penalize otherwise good uni-directional rules in terms of confidence. We denote confidence by ***Conf***, and define the confidence for a sentinel rule, $A \rightsquigarrow B$, as follows:

$$Conf_{A \rightsquigarrow B} = \frac{\sum_{IndRule_i \in MaxRule} ElimSupp(IndRule_i)}{SentSupp_{A \rightsquigarrow B}} \tag{2.10}$$

The minimum threshold for ***SentSupp*** is denoted β, and the minimum threshold for ***Conf*** is denoted γ. With these definitions, we say that a sentinel rule, $A \rightsquigarrow B$,

with an offset, o, and a warning period, w, exists in C when $SentSupp_{A \rightsquigarrow B} \geqq \beta$ and $Conf_{A \rightsquigarrow B} \geqq \gamma$. $\alpha, \beta, \gamma, o,$ and w are provided by the user, and typically set iteratively based on the user's experience.

Sentinel rule notation: To express sentinel rules with easy readability, we use $\rightsquigarrow$ to show that there is a sentinel rule between a source measure, A, and a target measure, B. In the case, where a bi-directional rule represents an inverted relationship between the source and the target measure, we add *inv* to the target measure. In the case where the rule is uni-directional, we add ▲ or ▼ to both the source and the target measure to express the direction of the sentinel rule.

In our running example, we limit ourselves to investigating whether sentinel rules exist between any of the source measures $M_1 ... M_3$ and the target measure M_4. We now need to compare the changes in $M_1 ... M_3$ to changes in M_4 at a later time. In this case, we choose the timeframe of 1 quarter again, meaning that warning period *w = 1 Quarter*. In Table 2.3, we show the comparison between the source measure indications and the target measure indication one quarter later. The measure M_4 is basically moved one line up -or as shown in Table 2.3; one quarter back. This means that all source measures for Asia changing 2007: Q2→Q3 as shown in the left column are now compared on the same line, within the same row, to the change on the target measure, M_4, for Asia changing 2007: Q3→Q4 and so on. The shift of M_4 shown in the row with data for the period one quarter earlier is denoted M_4'. Please note that since we are looking at changes between the periods selected on the time dimension, as noted earlier, we naturally get one less row for each geographical region, when we make the comparison across 1 quarter.

Based on Table 2.3, we count the support for each combination of indication changes, the indication rules, for each potential sentinel rule; in addition, we can count the support of the relationship overall, basically the support means counting all rows that do not have a "neutral" change on the source measure since we define

T: Time	D_2: Region	M_1: NBlgs	M_2: CstPrb	M_3: WHts	M_4': Rev
'07:Q1→Q2	Asia	neutral	M_2▼	M_3▲	M_4'▲
'07:Q2→Q3	Asia	M_1▼	M_2▼	M_3▲	M_4'▲
'07:Q1→Q2	EU	M_1▼	M_2▼	M_3▲	M_4'▲
'07:Q2→Q3	EU	M_1▼	M_2▲	M_3▲	M_4'▲
'07:Q1→Q2	USA	M_1▲	M_2▲	M_3▲	M_4'▼
'07:Q2→Q3	USA	M_1▲	neutral	M_3▲	M_4'▼

Table 2.3: Target and source measure comparison

indications as being either positive or negative. For example, we see summarized in Table 2.4(a), that the indication rule $M_1\blacktriangledown \rightarrow M_4'\blacktriangle$ is supported 3 times in the dataset shown in Table 2.3; we say that the indication rule $M_1\blacktriangledown \rightarrow M_4'\blacktriangle$ has a support of 3, and the sentinel rule $M_1 \rightsquigarrow M_4$ has a support of all indication rule combinations which in this case is 5. Table 2.4(a) through 2.4(c) lists the indication rules for each potential sentinel rule with their respective support (Formula (2.9)).

As mentioned earlier, the ideal sentinel rule describes changes bi-directionally so that it can "predict" both positive and negative changes on the target measure. However, the relationship also needs to be non-contradictory in order to be useful as a sentinel rule. To do this, we eliminate the indications that contradict each other as described in Formulae (2.6) and (2.7). In Table 2.4(b) we find the a uni-directional rule where the two contradicting indication rules have equal support, thus we disregard these indications completely (Formula (2.9)) and therefore $SentSupp_{M_2 \rightsquigarrow M_4}$=3. In Table 2.4(c) the contradiction elimination process does not eliminate both indication rules, it reduces the two indication rules to one and decreases *ElimSupp* (Formula (2.7)) in the calculation of confidence.

In order to identify the best sentinel rules, we set the thresholds $\beta = 3$ and $\gamma = 60\%$. Table 2.4(d) through 2.4(f) show the sentinel rules from our running example and their respective conformance to the thresholds we have set. As seen in Table 2.4(e) and 2.4(f), we end up having uni-directional sentinel rules, since the indication rules $M_2\blacktriangle \rightarrow M_4'\blacktriangle$ and $M_2\blacktriangle \rightarrow M_4'\blacktriangledown$, as shown in Table 2.4(b), contradict each other and have equal support. In addition, the indication rules $M_3\blacktriangle \rightarrow M_4'\blacktriangle$ and $M_3\blacktriangle \rightarrow M_4'\blacktriangledown$ contradict each other in Table 2.4(c). Of these, $M_3\blacktriangle \rightarrow M_4'\blacktriangle$ is strongest and "wins" the elimination process (Formula (2.8)) as seen in Table 2.4(f).

(a) $M_1 \rightsquigarrow M_4$

M_1	M_4'	*IndSupp*
$M_1\blacktriangle$	$M_4'\blacktriangle$	0
$M_1\blacktriangledown$	$M_4'\blacktriangledown$	0
$M_1\blacktriangle$	$M_4'\blacktriangledown$	2
$M_1\blacktriangledown$	$M_4'\blacktriangle$	3
$SentSupp_{M_1 \rightsquigarrow M_4}$ = 5		

(b) $M_2 \rightsquigarrow M_4$

M_2	M_4'	*IndSupp*
$M_2\blacktriangle$	$M_4'\blacktriangle$	1
$M_2\blacktriangledown$	$M_4'\blacktriangledown$	0
$M_2\blacktriangle$	$M_4'\blacktriangledown$	1
$M_2\blacktriangledown$	$M_4'\blacktriangle$	3
$SentSupp_{M_2 \rightsquigarrow M_4}$ = 3		

(c) $M_3 \rightsquigarrow M_4$

M_3	M_4'	*IndSupp*
$M_3\blacktriangle$	$M_4'\blacktriangle$	4
$M_3\blacktriangledown$	$M_4'\blacktriangledown$	0
$M_3\blacktriangle$	$M_4'\blacktriangledown$	2
$M_3\blacktriangledown$	$M_4'\blacktriangle$	0
$SentSupp_{M_3 \rightsquigarrow M_4}$ = 6		

(d) $M_1 \rightsquigarrow M_4$

M_1	M_4'	*ElimSupp*
$M_1\blacktriangle$	$M_4'\blacktriangledown$	2
$M_1\blacktriangledown$	$M_4'\blacktriangle$	3
$SentSupp_{M_1 \rightsquigarrow M_4}$ = 5		
$Conf_{M_1 \rightsquigarrow M_4} = \frac{5}{5}$ = 100%		
Conformance: ok		

(e) $M_2 \rightsquigarrow M_4$

M_2	M_4'	*ElimSupp*
$M_2\blacktriangledown$	$M_4'\blacktriangle$	3
$SentSupp_{M_2 \rightsquigarrow M_4}$ = 3		
$Conf_{M_2 \rightsquigarrow M_4} = \frac{3}{5}$ = 60%		
Conformance: ok		

(f) $M_3 \rightsquigarrow M_4$

M_3	M_4'	*ElimSupp*
$M_3\blacktriangle$	$M_4'\blacktriangle$	2
$SentSupp_{M_3 \rightsquigarrow M_4}$ = 6		
$Conf_{M_3 \rightsquigarrow M_4} = \frac{2}{6}$ = 33%		
Conformance: failed		

Table 2.4: Indication rule and sentinel rule support

Based on this example, we have now found that there are two sentinel rules that can provide our company with an early warning. If we monitor the changes to M_1, the number of negative blog entries, we will know one quarter in advance whether to expect an increase or a decrease in M_4 Revenue. If we monitor the number of times a customer contacts our company with a problem related to our products, M_2, we will know one quarter ahead whether to expect an increase in Revenue. This example demonstrates the usefulness of the sentinel concept, and the idea is that we attempt to place our sentinels as close to the external environment as possible and with as high reliability as possible. Using the notation defined earlier in this section, we can express the rules found in our running example as follows: $NBlgs \rightsquigarrow inv(Rev)$ and $CstPrb\blacktriangledown \rightsquigarrow Rev\blacktriangle$

2.3 The FindSentinels Algorithm

The following algorithm has been implemented in SQL on a Microsoft SQL Server 2005 as explained in Section 2.5. The actual SQL code can be found in Section 2.4. We assume without loss of generality that of the p measures in the dataset, C, $M_1 ... M_{p-1}$ are the source measures and M_p is the target measure.

Step 1 creates a temporary table where each unique value of (time dimension value) *t*, is sorted in ascending order and assigned an integer, *Id*, growing by 1 for each *t*. This temporary table will allow us to select values of *t* for comparison with a given distance in periods, regardless of the format of the period field, *t*, in the database. To optimize performance, we create an index on the period table. By joining 4 copies of each of the original dataset and the period table (one for each of the periods: $t, t + o, t + w$, and $t + w + o$), we create a Source-Target set (Formula (2.5)) and calculate indications (Formulae (2.3) and (2.4)) for our selected *p-1* source measures and one target measure. We calculate these indications for each cell (dimension combination) in the dataset, and return -1, 0, or 1 depending on whether the indication is negative, neutral or positive against the threshold α.

Step 2 counts the number of positive and negative indications on the source measure, and for each of these source measure indications, it summarizes the indications on the target measure. Since the indications are expressed as -1, 0 or 1, our contradiction elimination process can be carried out using sum.

Step 3 retrieves the potential rules from previous output, meaning that a source measure needs to have at least one indication with a consequential indication on the target measure, i.e., *ElimSupp*$<>0$. For each of these rules, we calculate the sum of the support of source measure indications, *SentSupp*, the sum of absolute indications on the target measure, *AbsElimSupp*, as well as *MaxElimSupp* which is max(*ElimSupp*). In addition, we calculate the *Direction* of the relationship between source and target measure where 1 is straightforward and -1 is inverted. The nature

of *Direction* also helps us eliminate orthogonal rules since these will always have *Direction=0*. This is true because an orthogonal relationship means that both positive and negative indications on the source measure leads to only one type of indication on the target measure. Finally, we calculate the number of indication rules, *IndRuleCount*, in the potential sentinel rule. This information is used to distinguish between bi- and uni-directional rules. Using this information, we can now identify the sentinel rules that comply with the criteria of $SentSupp >= \beta$ and $Conf >= \gamma$. In addition, we can use the values of *IndRuleCount*, *Direction*, and *MaxElimSupp* to describe the sentinel rule in accordance with our notation. We store the output in a table called *FinalResult*.

Algorithm FindSentinels
Input: A dataset, *C*, an offset, o, a warning period, w, a threshold for indications, α, a minimum *SentSupp* threshold, β, and a minimum *Conf* threshold, γ.
Output: Sentinel rules with their respective *SentSupp* and *Conf*.
Method: Sentinel rules are discovered as follows:

1. Scan the dataset C once and retrieve unique values of t into an indexed subset. Use the subset to reference each cell $(t, d_2, ... ,d_n) \in C$ with the corresponding cells for $\{t+o, t+w, t+w+o\} \in C$. Output a Source-Target set (Formula (2.5)) for each cell, $(t, d_2, ... ,d_n)$, where the indications (Formulae (2.3) and (2.4)) on source measures, $M_1...M_{p-1}$, are calculated using $\{t, t+o\}$ and the indications on target measure, M_p, is calculated using $\{t+w, t+w+o\}$.
2. For each positive and negative source measure indication, $M_i Ind$, in the output from Step 1, count the number of source measure indications as $IndSupp_i$ and sum the target measure indications as $ElimSupp_i$.
3. Retrieve from the output from Step 2, each source measure, $M_i \in M_1...M_{p-1}$, where *ElimSupp*$<>0$.
   For each of these source measures, calculate: *SentSupp*=sum(*IndSupp*),
   AbsElimSupp=sum|*ElimSupp*|, *MaxElimSupp*=max(*ElimSupp*),
   Direction=avg(sign($M_i Ind$)*sign(*ElimSupp*)), and *IndRuleCount* as the number of different indications (positive, negative). Output the rules where $SentSupp >= \beta$ and $Conf = \frac{AbsElimSupp}{SentSupp} >= \gamma$, use *IndRuleCount*=2 to identify bi-directional rules and *Direction* to describe whether the relationship is straight-forward or inverted. For uni-directional rules (*IndRuleCount*$= 1$) use the four combinations of *Direction* and sign(*MaxElimSupp*) to describe the relationship.

Upon execution of the algorithm, FindSentinels, with the dataset from our running example as *C*, we get the output table named *FinalResult* as seen in Table 2.5. We note that the result is similar to that of Tables 2.4(d) & 2.4(e), and we can easily recognize the sentinel rules: $NBlgs \rightsquigarrow inv(Rev)$ and $CstPrb\blacktriangledown \rightsquigarrow Rev\blacktriangle$

SentinelRule	SentSupp	Conf
NBlgs->inv(Rev)	5	100
CstPrb_dec->Rev_inc	3	100

Table 2.5: The *FinalResult* table

Computational Complexity: When we examine the individual statements of the algorithm, FindSentinels, we notice that the size of output for each step is at most as large as the input, thus the computational complexity will be dominated by the size of the input. The size of the input for Step 3 is much smaller than for previous statements and can thus be disregarded. The retrieve of unique *t* in Step 1 can be performed in $\mathcal{O}(n)$ using a hash-based algorithm [22], where *n* is the size of the dataset, *C*. The indexing can be done in time $\mathcal{O}(p\ log\ p)$ where *p* is the number of periods, and since $p << n$ this cost can be disregarded. The major cost in Step 1 is the 8-way join, and since $p << n$ this cost will be dominated by the cost of the join of 4 copies of *C*, and even this can be performed in time $\mathcal{O}(n)$ using hash-joins [22]. Since the number of source measures are a small constant, Step 2 can also be done in $\mathcal{O}(n)$ using hash-aggregation [22]. In summary, the whole algorithm can be done in time $\mathcal{O}(n)$, where *n* is the size of *C*, and the algorithm thus scales linearly.

2.4 SQL-Based Implementation

The following SQL is written for Microsoft SQL Server 2005 as explained in Section 2.3. We assume without loss of generality that of the p measures in the dataset, C, $M_1 ... M_{p-1}$ are the source measures and M_p is the target measure.

Statement 1 creates a temporary table called *Period* where each unique value of (time dimension value) *t*, is sorted in ascending order and assigned an integer, *Id*, growing by 1 for each *t*. This table will allow us to select values of *t* for comparison with a given distance in periods, regardless of the format of the period field, *t*, in the database. In Statement 2, we create an index on the period table to optimize performance of the following operations.

By joining 4 copies of each of the original dataset and the period table (one for each of the periods: $t, t+o, t+w$, and $t+w+o$) in Statement 3, we create a table called *Result3* which contains the indications on the source measures as a result of the differences in measure values from period *id* to period *id+o*. In addition, we calculate the indications on the target measure from period *id+w* to period *id+w+o*. We calculate these indications for each cell (dimension combination) in the dataset. The custom function, *Ind*, handles the Formulae (2.3) and (2.4) and returns -1, 0, or 1 depending on whether the indication is negative, neutral or positive against the threshold α. This statement handles the shifting of facts, Formulae (2.1) and (2.2), and produces a Source-Target set (Formula (2.5) for our selected *p-1* source measures and one target measure. After executing Statement 3, we have a table which contains the same information as Table 2.3 in our running example with indications expressed as -1, 0 or 1.

Statement 4 counts the number of positive and negative indications on the source measure, and for each of these source measure indications, it summarizes the indi-

cations on the target measure. The output is stored in *Result4*. Since the indications are expressed as -1, 0 or 1, our contradiction elimination process can be carried out simply by this summation.

Statement 5 retrieves the potential rules from *Result4*, meaning that a source measure needs to have at least one indication with a consequential indication on the target measure, i.e., *ElimSupp*$<>0$. For each of these rules, we calculate the sum of the support of source measure indications, *SentSupp*, the numeric sum of indications on the target measure, *AbsElimSupp*, as well as *MaxElimSupp* which is max(*ElimSupp*). In addition, Statement 5 calculates the *Direction* of the relationship between source and target measure where 1 is straightforward and -1 is inverted. The nature of *Direction* also helps us eliminate orthogonal rules since these will always have *Direction=0*. This is true because an orthogonal relationship means that both positive and negative indications on the source measure leads to only one type of indication on the target measure. Finally, Statement 5 calculates the number of indication rules, *IndRuleCount*, in the potential sentinel rule. This information is used to distinguish between bi- and uni-directional rules. The output of Statement 5 is stored in *Result5*.

Using *Result5* in Statement 6, we identify the sentinel rules that comply with the criteria of $SentSupp >= \beta$ and $Conf >= \gamma$. We use the values of *IndRuleCount*, *Direction*, and *MaxElimSupp* to describe the sentinel rule in accordance with our notation. The output is stored in *FinalResult*.

SQL Statements for FindSentinels($C, o, w, \alpha, \beta, \gamma$)

```
(1) SELECT T, ROW_NUMBER() OVER (ORDER BY T) AS Id
     INTO Period FROM (SELECT DISTINCT T FROM C) AS subselect

(2) CREATE INDEX PeriodIndex ON Period(T)

(3) SELECT a.T, a.D2, ... , a.Dn,
           Ind((b.M1-a.M1)/a.M1,α) AS M1ind,
           Ind((b.M2-a.M2)/a.M1,α) AS M2ind,
           ...
           Ind((d.Mp-c.Mp)/c.Mp,α) AS TMind
     INTO Result3
     FROM C a, C b, C c, C d,
          Period pa, Period pb, Period pc, Period pd,
     WHERE a.t=pa.t AND b.t=pb.t AND c.t=pc.t AND d.t=pd.t AND
           a.D2=b.D2 AND a.D2=c.D2 AND a.D2=d.D2 AND
           ...
           a.Dn=b.Dn AND a.Dn=c.Dn AND a.Dn=d.Dn AND
           pb.id=pa.id+o AND pc.id=pa.id+w pd.id=pa.id+w+o

Custom function Ind is implemented as follows for source and target measure:
    SIGN(ROUND(((b.Mi-a.Mi)/a.Mi)*100/α, 0, 1)) AS Miind
    SIGN(ROUND(((d.Mp-c.Mp)/c.Mp)*100/α, 0, 1)) AS TMind

(4) SELECT sourcemeasure, Ind, COUNT(*) AS IndSupp, SUM(TMind) AS ElimSupp
     INTO Result4
     FROM (SELECT "M1" AS SourceMeasure, M1ind AS Ind, TMind FROM Result3
     UNION ALL SELECT "M2" AS SourceMeasure, M2ind AS Ind, TMind FROM Result3
```

```
    ...
    UNION ALL SELECT "M_{p-1}" AS SourceMeasure, M_{p-1}ind AS Ind, TMind
    FROM Result3) AS subselect
    WHERE Ind<>0
    GROUP BY SourceMeasure, Ind

(5) SELECT SourceMeasure, SUM(IndSupp) AS SentSupp,
           SUM(ABS(ElimSupp)) AS ElimSupp, MAX(ElimSupp) AS MaxElimSupp,
           AVG(SIGN(Ind)*SIGN(ElimSupp)) AS Direction,
           COUNT(*) AS IndRuleCount
    INTO Result5
    FROM Result4
    WHERE ElimSupp<>0
    GROUP BY SourceMeasure
(6) SELECT SentinelRule, SentSupp, Conf INTO FinalResult FROM
        (SELECT SourceMeasure+'->inv(target)' AS SentinelRule,
           SentSupp, ElimSupp*100/SentSupp AS Conf FROM Result5
           WHERE Direction<0 and IndRuleCount=2
        UNION ALL SELECT SourceMeasure+'->target' AS SentinelRule,
           SentSupp, ElimSupp*100/SentSupp AS Conf FROM Result5
           WHERE Direction>0 and IndRuleCount=2
        UNION ALL SELECT SourceMeasure+'_inc->target_dec' AS SentinelRule,
           SentSupp, ElimSupp*100/SentSupp AS Conf FROM Result5
           WHERE Direction<0 and MaxElimSupp<0 and IndRuleCount=1
        UNION ALL SELECT SourceMeasure+'_dec->target_inc' AS SentinelRule,
           SentSupp, ElimSupp*100/SentSupp AS Conf FROM Result5
           WHERE Direction<0 and MaxElimSupp>0 and IndRuleCount=1
        UNION ALL SELECT SourceMeasure+'_dec->target_dec' AS SentinelRule,
           SentSupp, ElimSupp*100/SentSupp AS Conf FROM Result5
           WHERE Direction>0 and MaxElimSupp<0 and IndRuleCount=1
        UNION ALL SELECT SourceMeasure+'_inc->target_inc' AS SentinelRule,
           SentSupp, ElimSupp*100/SentSupp AS Conf FROM Result5
           WHERE Direction>0 and MaxElimSupp>0 and IndRuleCount=1
        ) as subselect
    WHERE SentSupp>=β and Conf>=γ
```

2.5 Experiments

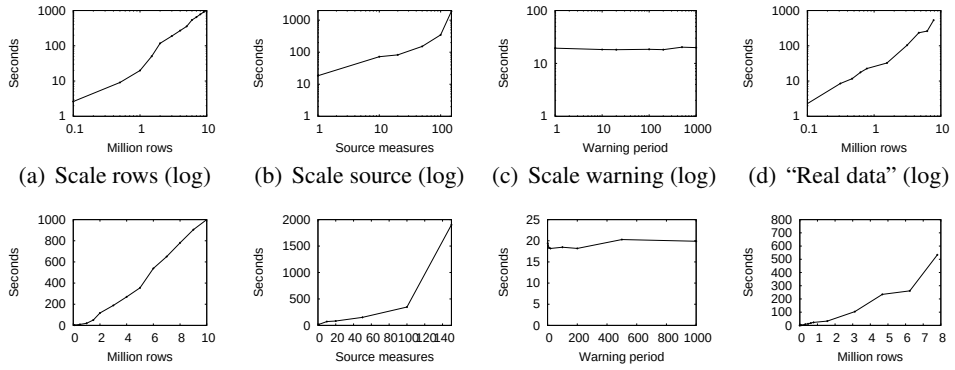
Setup: The experimental evaluation was conducted on an Intel Core2 Quad CPU (Q6600) 2.40GHz PC server with 4GB RAM and 1 500GB disk (7,200 RPM) run-

ning a 32Bit version of Microsoft SQL Server 2005 (MS SQL) on Windows Vista Business, Service Pack 1. The recovery model on MS SQL was set to "simple". In addition, MS SQL is restarted and the database and logfile space are shrunk before each run.

We use two ranges of datasets for the experiments, a range of synthetic datasets and a range of real-world datasets. The synthetic datasets closely resemble our running example, i.e., there are three regions and one product that are iterated over a variable number of periods and variable number of source measures to produce a dataset of the desired size in rows and source measures. The synthetic datasets produce the same sentinel rules as output as our running example when subjected to our SQL implementation. The synthetic datasets with three source measures and one target measure, similar to our running example, ranges from 100,000 to 10,000,000 rows with 1,000,000 row intervals from 2,000,000 to 10,000,000 rows, and includes datasets with 500,000 and 1,500,000 rows in order to monitor the behavior of the SQL implementation more closely at low data volumes. In addition, we generate datasets with 1,000,000 rows and with 1, 10, 20, 50, 100 and 150 source measures and one target measure.

The real-world datasets are produced based on a dataset from a medium-sized Danish company with one location and one product, the original dataset contains 78 months (6.5 years) of monthly aggregated measures of *Website hits*, *Support cases*, *New customers* and *Revenue*. Descendants of this dataset are produced by adding the original dataset to itself the number of times needed to get the desired number of rows, but each time the dataset is added, it is done with a new product number. By doing this, we end up having a significant amount of data with real-world patterns. Using this approach, all real-world datasets have a number of rows in multiples of 78, namely: 78; 780; 7,800; 78,000; 312,000; 468,000; 624,000; 780,000; 1,560,000; 3,120,000; 4,680,000; 6,240,000 and 7,800,000 rows. The results presented represent the average time for 5 runs.

Synthetic data - scaling number of rows: We run the SQL implementation against all the synthetic datasets with three source and one target measure. We have plotted the results in both logarithmic scale, Figure 2.1(a), and linear scale, Figure 2.1(e), in order to get a better impression of the scaling at both the low end and the high end. We notice that the SQL implementation scales linearly as expected based on our assessment of the $\mathcal{O}(n)$ computational complexity. We attribute the jump in process time that occurs between the experiments on 5,000,000 and 6,000,000 rows, to the DBMS' inability to handle everything in memory thus requiring more disk I/O. A similar factor is seen between the experiments on 1,500,000 and 2,000,000 rows.



(a) Scale rows (log) (b) Scale source (log) (c) Scale warning (log) (d) "Real data" (log)

(e) Scale rows (f) Scale source (g) Scale warning (h) "Real data"

Figure 2.1: Performance results

Synthetic data - scaling source measures: The SQL implementation is run against the synthetic datasets with a varying number of source measures and 1,000,000 rows. The results are plotted in Figures 2.1(b) and 2.1(f). We observe a linear behavior in computation time until we reach more than 100 source measures where process time increases drastically. However, 100 measures is a very high number in a normal OLAP cube, since real-world implementations in our experience typically have a sum of source and target measures less than 50.

Synthetic data - scaling warning period: In this experiment, we vary the warning period, w, with the values: 1, 10, 20, 100, 200, 500, 1000 on a dataset with 1,000,000 rows. Results are plotted in Figures 2.1(c) and 2.1(g). As expected, the SQL implementation is almost unaffected by a variation in warning period, the process time remains close to constant because the intermediate results of the SQL solution's steps are about the same size.

Real-world data - scaling number of rows: Changing now to real-world datasets, we run the SQL implementation on the real-world datasets with varying number of rows. The results are plotted in Figures 2.1(d) and 2.1(h). In general, we observe a behavior close to linear, but the jump in process time due to a change from memory to disk, appear different from what we observed in the experiments on synthetic data. We attribute the variance to the fact that the joins behave slightly different since the period field has lower cardinality and the product field has higher cardinality in the real-world datasets compared to the synthetic datasets. We acknowledge, however, that overall the SQL implementation scales linearly on real-data; additionally, we notice that the performance on real-world data is faster than the performance on the synthetic data with a factor of 1.4 at 7,800,000 rows.

Qualitative Experiment: Other than a performance assessment of our SQL implementation, we conduct an experiment by allowing the *o* and *w* values to vary on the original real-world dataset. Our inspiration is a business case in which we want to find the best sentinel rules in terms of *SentSupp* and *Conf*. We vary the granularity of the offset period, *o*, to be either month, quarter or year. We vary the warning period, *w*, to be of same granularity as the offset, and to start anywhere in between the period defined by the offset and 12 months prior to the period defined by the offset. When running this experiment, we found an interesting sentinel rule with *SentSupp* of 33 out of 78 input rows and 79.5% *Conf* that told us that if the number of Website hits goes up one year, the revenue will go up the following year. Although this rule is not a surprise, since we are dealing with a company in growth, we notice that our solution has discovered a sentinel rule by autonomously selecting *o* and *w* on real data, and the rule is exactly of the type and on the level of detail that we are looking for.

Experiment Summary: From the experiments we validate that our SQL implementation does indeed have $\mathcal{O}(n)$ computational complexity, and that it scales linearly to 10 million rows of aggregate data. We recall that a real-world dataset can easily be as small as 78 rows. A real-world scenario can, however, increase in complexity when the cardinality increases on the dimensions involved. From these experiments, we would expect that good performance can be gained for a real-world organization, e.g., on 6.5 years of data aggregated monthly for a company with 100 geographical locations and 1,000 products gives 7,800,000 rows. Furthermore, we found that even though a brute-force approach was applied in fitting *o* and *w* to get the best sentinel rules, such an approach can realistically be used on a real dataset to provide relevant results.

2.6 Related Work

For decades, focus has been on immediate data warehousing challenges of obtaining and storing data for business purposes, and making it accessible with high performance [61]. OLAP applications have emerged, but with the exception of Kimball's "Analytical Cycle" [30], there has been little theory such as CALM [36] about how to combine specific BI technologies to meet the management challenge of turning data into valuable decisions.

The idea that some actions or incidents are interlinked has been well explored in *association rules* [2]. The traditional case study of association rules has been basket-type association rules, and significant effort has been put into optimizing the initially proposed Apriori algorithm [3], including its extension to exploit the performance of a parallel shared-nothing multiprocessor system [6]. Improvements in performance as well as lower memory usage compared to the Apriori algorithm has been gained

by growing a compressed frequent pattern tree in memory [25], and improvements in the selection process of thresholds that determine the interestingness of rules has also been introduced [58].

Sequential pattern mining introduces a sequence in which actions or incidents take place, with the intention of predicting one action or incident based on knowing another one. This adds to the complexity of association rules which makes the Apriori approach even more costly [5], thus new approaches to improving the performance of mining sequential patterns have emerged [24,48,49,57]. Another aspect of sequential pattern mining has been the various approaches to handling the period of the sequence, e.g., involving multiple time granularities [9] or allowing for partial periodicity of patterns [23]. The focus on user control of the sequential pattern mining process in terms of applying constraints [21], and in a querying-type approach [20] has also been explored. The introduction of multi-dimensional databases has given rise to *multi-dimensional pattern mining* [51] which applies the same techniques to more dimensions than just one.

In general, association rule mining seeks to find co-occurrence patterns within *absolute data values*, whereas our solution works on the *relative changes in data*. In addition, association rule mining typically works on *categorical data*, i.e., dimension values, whereas our solution works on *numerical data* such as measure values. Sequential pattern mining allows a time period to pass between the premise and the consequent in the rule, but it remains focused on co-occurrence patterns within absolute data values for categorical data. Furthermore, our solution generates rules at the *schema level*, as opposed to the *data level*, using a contradiction elimination process. The combination of schema-level rules based on relative changes in data allows us to generate fewer, more general, rules that cannot be found with neither association rules nor sequential pattern mining. In Section 2.7 we demonstrate why sequential pattern mining does not find any meaningful rules in our running example presented in Section 2.2.

Other approaches to interpreting the behavior of data sequences are various regression [4] and correlation [26,60] techniques which attempt to describe a functional relationship between one measure and another. In comparison, we can say that sentinel rules are a set of *"micro-predictions"* that are complementary to regression and correlation techniques. Sentinel rules are useful for discovering strong relationships between a smaller subset within a dataset, and thus they are useful for detecting warnings whenever changes (that would otherwise go unnoticed) in a relevant source measure occur. In addition, regression and correlation techniques do not support uni-directional relationships such as our solution. Regression and correlation based techniques, on the other hand, are useful for describing the overall trends within a dataset. In Section 2.8, we specifically demonstrate using a concrete, realistic example how correlation tend to *blind the user by the average*.

Finally, the work on similarity search in timeseries databases [4] attempt to describe periods in which one measure behaves similar to another. This work is different from sentinel rules since it does not generate schema-level rules (or rules at all), furthermore it does not allow the description of a uni-directional relationships.

2.7 Sentinels Rules vs. Sequential Pattern Mining

Sequential pattern mining identifies patterns of items that occur with a certain frequency in a dataset of transactions grouped in sequences, i.e., a sequence could look like this: $\{(ab)c\}$, where *(ab)* and c are transactions. The support of patterns such as "item *a* leads to item *c*" are found by counting the number of times a transaction containing *a* is followed by a transaction containing c.

To exemplify the difference between our solution and sequential pattern mining; specifically the difference between working on absolute data *values* and relative data *changes*, we can simply subject our running example dataset in Table 2.1 to sequential pattern mining. Since *all* values in this dataset are *unique*, we will only find patterns with an absolute support of only 1 and confidence of 100%, as follows:

"NBlgs=20 leads to Rev=9000"
"CstPrb=50 leads to Rev=9000"
"WHts=1000 leads to Rev=9000"
"NBlgs=20 and CstPrb=50 leads to Rev=9000"
"NBlgs=20 and WHts=1000 leads to Rev=9000"
"NBlgs=20 and CstPrb=50 and WHts=1000 leads to Rev=9000"
...

The rules above above have been generated using only the first line in the dataset in Table 2.1, meaning that the complete set would generate 12x6=72 rules. If we think of a 1,000 record dataset with the same properties as Table 2.1, such a dataset would result in 6,000 rules. In other words, since the data-level rules generated by sequential pattern mining are *value specific*, as opposed to relative change "events" at the schema level, we get a large output in which we do not find any significant patterns. Thus no meaningful, high-level rules can be found by sequential pattern mining directly.

2.8 Sentinels Rules vs. Correlation Techniques

If we compare sentinel rules with correlation techniques, one could intuitively think that correlations would be able to detect exactly the same relationships as sentinel

rules, at least when comparing correlations to bi-directional sentinel rules. However, when subjecting data to correlation techniques [26, 60], we do not find the same explicit relationships as we do with sentinel rules.

The reason that correlation techniques gives different results compared to our sentinel rules, lies in the fact, that correlation techniques are focused on identifying relationships with the minimal Euclidean distance between pairs, this means that there is a tendency to favor "smooth" data series with few outliers. On the other hand, the intention of sentinel rules is to give a user an early warning, and from an operational perspective we can even say that we are particularly interested in the source measure changes (perhaps unusual/"outlier") that appear to have consequential change effect on the target measure. This difference means (from a sentinel perspective) that correlation techniques "blind us by the average" when our goal is to get an early warning.

Tables 2.6(a)–2.6(c) exemplify these differences by describing the relationship between a source measure, A, and a target measure, B, with both sentinel rules and correlation. The data series in Tables 2.6(a)–2.6(c) have been pre-processed so that each line represents a combination of A for time period, t, and B for time period, $t + w$ ($w = 1$). The distance in time between the lines is o. We operate with the same values for α, β, and γ as in our running example, specifically: $\alpha = 10\%$, $\beta = 3$, and $\gamma = 60\%$. For correlation, we say that it needs to account for more than 50% of the variation (correlation-coefficient^2 > 0.5) between the series in order to be relevant. In fact, it does not even make sense to consider correlations that account for 50% or less, since such correlations would be less precise than simply flipping a coin to

(a) $A \rightsquigarrow inv(B)$

A	B	Indication
99	1000	
89	1100	$A\blacktriangledown \rightarrow B\blacktriangle$
90	1001	
98	1015	
113	900	$A\blacktriangle \rightarrow B\blacktriangledown$
101	1025	$A\blacktriangledown \rightarrow B\blacktriangle$
108	1100	
105	1090	
109	1040	
90	1145	$A\blacktriangledown \rightarrow B\blacktriangle$
Sentinel rule:		
$SentSupp_{A \rightsquigarrow inv(B)}$ = 4		
$Conf_{A \rightsquigarrow inv(B)}$ = 100%		
Correlation:		
Coefficient = -0.4307		
Accounts for = 18.55%		

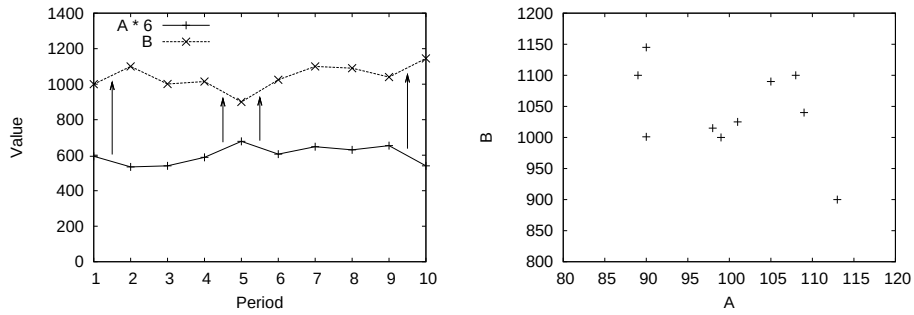
(b) $A\blacktriangledown \rightsquigarrow B\blacktriangle$

A	B	Indication
500	1000	
400	1200	$A\blacktriangledown \rightarrow B\blacktriangle$
363	1095	
399	1200	
350	1400	$A\blacktriangledown \rightarrow B\blacktriangle$
316	1265	
323	1285	
355	1410	
300	1600	$A\blacktriangledown \rightarrow B\blacktriangle$
329	1755	
Sentinel rule:		
$SentSupp_{A\blacktriangledown \rightsquigarrow B\blacktriangle}$ = 3		
$Conf_{A\blacktriangledown \rightsquigarrow B\blacktriangle}$ = 100%		
Correlation:		
Coefficient = -0.6919		
Accounts for = 47.87%		

(c) $A\blacktriangledown \rightsquigarrow B\blacktriangle$

A	B	Indication
100	1000	
90	1100	$A\blacktriangledown \rightarrow B\blacktriangle$
81	1210	$A\blacktriangledown \rightarrow B\blacktriangle$
73	1331	$A\blacktriangledown \rightarrow B\blacktriangle$
66	1464	$A\blacktriangledown \rightarrow B\blacktriangle$
59	1611	$A\blacktriangledown \rightarrow B\blacktriangle$
53	1772	$A\blacktriangledown \rightarrow B\blacktriangle$
48	1949	$A\blacktriangledown \rightarrow B\blacktriangle$
43	2144	$A\blacktriangledown \rightarrow B\blacktriangle$
39	2358	$A\blacktriangledown \rightarrow B\blacktriangle$
Sentinel rule:		
$SentSupp_{A\blacktriangledown \rightsquigarrow B\blacktriangle}$ = 9		
$Conf_{A\blacktriangledown \rightsquigarrow B\blacktriangle}$ = 100%		
Correlation:		
Coefficient = -0.9688		
Accounts for = 93.85%		

Table 2.6: Example relationships.



(a) Linear Plot (b) Scatter Plot

Figure 2.2: Graphs based on data from Table 2.6(a).

decide whether target measure, B, will go up or down. We can now read the individual indications directly, and by correlating the two data series, we get a lagged correlation with the same warning period as the sentinel rules. Table 2.6(a) shows a bi-directional sentinel rule, $A \rightsquigarrow inv(B)$, that is **not** discovered using correlation techniques, . Table 2.6(b) shows a uni-directional sentinel rule, $A\blacktriangledown \rightsquigarrow B\blacktriangle$, that is **not** discovered using correlation techniques. In . Table 2.6(c), correlation and sentinel rules both find a uni-directional rule, $A\blacktriangledown \rightsquigarrow B\blacktriangle$.

The data from . Table 2.6(a) is plotted in Fig. 2.2(a), note again that the top line (B) is shifted $w = 1$ period(s) backwards. We note the visible changes for A and B, where A has been scaled by a factor of 6 for better visibility. The changes that leads to sentinel rules have been highlighted by an arrow from A to B. When looking at Fig. 2.2(a) it seems clear that there is a relationship between the changes. However, when displaying the same data in a scatter plot, Fig. 2.2(b), to test the correlation visually, we find that the relationship between A and B seems to be rather chaotic or at least non-linear. This is the reason that we get a very poor correlation, whereas on the other hand, we find a strong sentinel rule based on four indications.

Following these examples, we confirm the differences between sentinel rules and correlation thecniques by applying correlation techniques [26, 60] to our running example in Table 2.1. Correlation techniques find lagged correlations between all source measures and the target measure, but the only correlation that accounts for more than 50% of the variation, is identified between *WHts* and *Rev* which is in contrast to the bi- and uni-directional sentinel rules between *NBlgs* and *Rev*, and between *CstPrb* and *Rev*, which we found with our solution.

In summary, we could say that sentinel rules are a set of *"micro-predictions"* that are complementary to correlation techniques. Sentinel rules are useful for discovering strong relationships between a smaller subset within a dataset, and thus they are useful for detecting warnings whenever changes (that would otherwise go unnoticed)

in a relevant source measure occur. Correlation techniques, on the other hand, are useful for describing the overall trends within a dataset.

2.9 Conclusion and Future Work

We have proposed a novel approach for discovering so-called sentinel rules in a multi-dimensional database for business intelligence. The sentinel rules were generated at schema level, which means that they are more general and cleansed for contradictions, and thus easy to interpret. These sentinel rules can be used to expand the window of opportunity for an organization to act based on changes in the environment in which it operates. We demonstrated how an implementation in SQL could be done, and we showed that it scales linearly on large volumes of both synthetic and real-world data. We also demonstrated that sentinel rules with relevance for decision making can be extracted from real-world data. In this context, we proved the possibility of automatic fitting of both warning and observation periods. With regards to novelty, we specifically demonstrated that sentinel rules are different from sequential pattern mining, since sentinel rules operate at the schema level and use a contradiction elimination process to generate fewer, more general rules. Furthermore, we found sentinel rules to be complementary to correlation techniques, in that they could discover strong relationships between a smaller subset within a dataset; a relationship that would otherwise be "hidden in the average" using correlation techniques alone.

There are several perspectives for future work. First, an idea would be to use the SQL implementation described in this chapter as a baseline for new improved algorithms in terms of performance. Secondly, the ability to automatically fit $\alpha, \beta, \gamma, o,$ and w on large volumes of data and different granularities should be explored. Third, it would make sense to seek for rules where multiple source measures are combined into a sentinel rule. Fourth, a natural development would also be to seek sentinel rules for multiple target measures at the same time to improve overall performance. Fifth, an idea could be to improve the solution in the multi-dimensional environment by allowing the sentinel rule mining to fit the aggregation level on dimensions automatically as well as automatically select the location and shape of the data area where the sentinel rules best apply.

From a business and decision-making stand point, more effort should be put into automatically pruning the sentinel rules found e.g., by assessing their interrelated relevance. Additionally, the degree of balance between the positive and the negative indications behind the sentinel rule, or to which degree rules are irrelevant based on orthogonal relationships between the source and the target measure, should be further explored in order for it to be automated.

Chapter 3

Efficient Discovery of Generalized Sentinel Rules

This chapter proposes the concept of *generalized sentinel rules* (sentinels) and presents an algorithm for their discovery. Sentinels represent *schema level* relationships between *changes over time* in certain measures in a multi-dimensional data cube. Sentinels notify users based on previous observations, e.g., that revenue might drop within two months if an increase in customer problems combined with a decrease in website traffic is observed. If the vice versa also holds, we have a *bi-directional* sentinel, which has a higher chance of being causal rather than coincidental. We significantly extend prior work to combine multiple measures into better sentinels as well as auto-fitting the best warning period. We introduce two novel quality measures, *Balance* and *Score*, that are used for selecting the best sentinels. We introduce an efficient algorithm incorporating novel optimization techniques. The algorithm is efficient and scales to very large datasets, which is verified by extensive experiments on both real and synthetic data. Moreover, we are able to discover strong and useful sentinels that could not be found when using sequential pattern mining or correlation techniques.

3.1 Introduction

The Computer Aided Leadership and Management (CALM) concept copes with the challenges facing managers that operate in a world of chaos due to the globalization of commerce and connectivity [36]; in this chaotic world, the ability to continuously *react* is far more crucial for success than the ability to *long-term forecast*. The idea in

CALM is to take the Observation-Orientation-Decision-Action (OODA) loop (originally pioneered by "Top Gun" fighter pilot John Boyd in the 1950s), and integrate business intelligence (BI) technologies to drastically increase the speed with which a user in an organization cycles through the OODA loop. One way to improve the speed from observation to action is to expand the "time-horizon" by providing the user of a BI system with warnings based on "micro-predictions" of changes to an important measure, often called a Key Performance Indicator (KPI). A *generalized sentinel rule* (*sentinel* for short) is a causal relationship where changes in one or *multiple source measures*, are followed by changes to a *target measure* (typically a KPI), within a given time period, referred to as the *warning period*. We attribute higher quality to bi-directional sentinels that can predict changes in both directions, since such a relationship intuitively is less likely to be coincidental. An example of a sentinel for a company could be: "IF Number of Customer Problems go up and Website Traffic goes down THEN Revenue goes down within two months AND IF Number of Customer Problems go down and Website Traffic goes up THEN Revenue goes up within two months". Such a rule will allow a BI system to notify a user to take corrective action once there is an occurrence of, e.g., "Customer Problems go up and Website Traffic goes down", since he knows, based on the "micro-prediction" of the rule, that Revenue, with the probability stated by the rule's confidence, will go down in two months if no action is taken. In Section 3.4 we describe an example where a valuable, and not so obvious, sentinel was uncovered.

Compared to prior art, sentinels are mined on the measures and dimensions of multiple cubes in an OLAP database, as opposed to the "flat file" formats used by most traditional data mining methods. Sentinels find rules that would be impossible to detect using traditional techniques, since sentinels operate on *data changes* at the *schema level* as opposed to absolute *data values* at the *data level* such as association rules [2] and sequential patterns typically do [5]. This means that our solution works on *numerical data* such as measure values, whereas association rules and sequential patterns work on *categorical data*, i.e., dimension values. In [38] we specifically provide a concrete, realistic example where nothing useful is found using these techniques, while sentinel mining *do* find meaningful rules. In addition, *bi-directional* sentinels are *stronger* than both association rules and sequential patterns since such relationships have a greater chance of being causal rather than coincidental. The schema level nature of sentinels gives rise to the table of combinations (TC) and the reduced pattern growth (RPG) optimization (see Section 3.3), and such optimizations can therefore not be offered by sequential pattern mining or other known optimizations for simpler "market basket"-type data such as [11]. In addition to the TC and RPG optimizations, the auto-fitting of the warning period, and the ability to combine source measures into better sentinel rules, adds to the distance between our solution and optimizations offered in prior art such as [3, 24, 46, 48, 49, 57, 59].

Gradual rule mining [10] is a process much like association rules, where the categorical data are created by mapping numerical data to fuzzy partitions, and thus this technique works on numerical data similar to our solution. However, similar to association rules and sequential patterns, gradual rule mining does not have the schema level property of sentinels that allows our solution to create the strong bi-directional rules. Moreover, the primary objective of gradual rules is to describe the absolute values of a measure, whereas our solution operates on *changes* in the measure values. Therefore, for similar reasons as mentioned above, gradual rule mining does not have the ability to use the TC and RPG optimizations, and neither does it have the ability to auto-fit a warning period for a given rule.

Other approaches to interpreting the behavior of data sequences are various regression [4] and correlation [26, 60] techniques which attempt to describe a functional relationship between one measure and another. Similar to gradual rules, these techniques are also concerned with the absolute values of a measure, as opposed to sentinels that are based on changes in the measure values. With regards to the output, sentinels are more *specific "micro-predictions"*, and are thus complementary to these techniques. Sentinels are useful for discovering strong relationships between a smaller subset within a dataset as explained in Chapter 2, and thus they are useful for detecting warnings whenever changes (that would otherwise go unnoticed) in a relevant source measure occur.

The novel contributions in this chapter are as follows: First we generalize the concept of sentinel rules from previous work into generalized sentinel rules, that allow multiple source measures to be combined in sentinels, and that can facilitate auto-fitting of the best warning period. In this context, we define two new qualitative measures for sentinels, namely: *Balance* and *Score*, and we expand the previous notation to support our generalization. Secondly, we present an algorithm for sentinel discovery that can combine multiple source measures and auto-fit the optimal warning period within a given range. In the algorithm, we introduce and define the optimization technique called *Reduced Pattern Growth*, and we apply *Hill Climbing* for further optimization. In addition, our algorithm uses a so-called *Table of Combinations* that efficiently supports these optimization techniques. Third, we assess the computational complexity and conduct extensive experiments to validate our complexity assessment, and we verify that our optimized algorithm scales well on large volumes of real-world and synthetic data.

The remainder of the chapter is structured as follows: Section 3.2 presents the formal definition, Section 3.3 presents the new SentHiRPG algorithm and its implementation. Section 3.4 presents a scalability study, and Section 3.5 presents our conclusions and proposals for future work.

3.2 Problem Definition

Running Example: We imagine having a company that sells products world-wide, and that we, in addition to the traditional financial measures such as revenue, *Rev*, have been monitoring the environment outside our organization and collected that information in three measures. The measure *NBlgs* represents the number of times an entry is written on a blog where a user is venting a negative opinion about our company or products. The measure *CstPrb* represents the number of times a customer contacts our company with a problem related to our products. The measure *WHts* represents the number of human hits on our website. We want to investigate whether we can use changes on any of the *external* measures (*NBlgs*, *CstPrb*, *WHts*) to predict a future change on the *internal measure* (*Rev*). To generalize our terminology, we call the external measures *NBlgs*, *CstPrb*, and *WHts source measures* and the internal measure, *Rev*, the *target measure*.

In Table 3.1 we see two subsets from our database, the source measures representing the external environment have been extracted for January to October 2008, and the target measure has been extracted for February to November 2008. For both source and target measures we have calculated the cases where a measure changes 10% or more, either up or down, from one month to another.

(a) Source

Month	NBlgs	CstPrb	WHts	Indications
2008-Jan	80	310	1227	
2008-Feb	89	390	1101	$NBlgs$▲,$CstPrb$▲,$WHts$▼
2008-Mar	90	363	1150	
2008-Apr	99	399	987	$NBlgs$▲,$WHts$▼
2008-May	113	440	888	$NBlgs$▲,$CstPrb$▲,$WHts$▼
2008-Jun	101	297	1147	$NBlgs$▼,$CstPrb$▼,$WHts$▲
2008-Jul	115	323	1003	$NBlgs$▲,$WHts$▼
2008-Aug	105	355	999	
2008-Sep	93	294	993	$NBlgs$▼,$CstPrb$▼
2008-Oct	100	264	1110	$CstPrb$▼,$WHts$▲

(b) Target

Month	Rev	Indications
2008-Feb	1020	
2008-Mar	911	Rev▼
2008-Apr	1001	
2008-May	1015	
2008-Jun	900	Rev▼
2008-Jul	1025	Rev▲
2008-Aug	1100	
2008-Sep	1090	
2008-Oct	970	Rev▼
2008-Nov	1150	Rev▲

Table 3.1: Source and target measure data with indications

Formal Definition: Let C be a multi-dimensional *cube* containing a set of *facts*, $C = \{(d_1, d_2, ..., d_n, m_1, m_2, ..., m_p)\}$. The dimension values, $d_1, d_2, ..., d_n$, belong to the *dimensions* $D_1, D_2, ..., D_n$, and we refer to the "dimension part" of a fact, $(d_1, d_2, ..., d_n)$, as a *cell*. We say that a cell belongs to C, denoted by $(d_1, d_2, ..., d_n) \in C$, when a fact $(d_1, d_2, ..., d_n, m_1, m_2, ..., m_p) \in C$ exists. We say that a *measure value*, m_i, is the result of a partial function, $M_i : D_1 \times D_2 \times ... \times D_n \hookrightarrow \Re$, denoted by, $M_i(d_1, d_2, ..., d_n) = m_i$, if $(d_1, d_2, ..., d_n) \in C$ and $1 \leqq i \leqq p$. We assume, without loss of generality, that there is only one time dimension, T, in C,

and that $T = D_1$, and subsequently $t = d_1$. In addition, we assume that measures $M_1, ..., M_{p-1}$ are source measures, and that measure M_p is the target measure. An *indication*, Ind_i, tells us whether a measure, M_i, changes by at least α over a period, o. We define $Ind_i(C, t, o, d_2, d_3, ..., d_n)$ as shown in Formula 3.1.

$$\text{if } (t, d_2, d_3, ..., d_n) \in C \wedge (t+o, d_2, d_3, ..., d_n) \in C \text{ then } Ind_i(C, t, o, d_2, d_3, ..., d_n) = \begin{cases} M_i\blacktriangle \text{ if } \frac{M_i(t+o, d_2, d_3, ..., d_n) - M_i(t, d_2, d_3, ..., d_n)}{M_i(t, d_2, d_3, ..., d_n)} \geqq \alpha \\ M_i\blacktriangledown \text{ if } \frac{M_i(t+o, d_2, d_3, ..., d_n) - M_i(t, d_2, d_3, ..., d_n)}{M_i(t, d_2, d_3, ..., d_n)} \leqq -\alpha \\ M_i \blacktriangleright \quad \text{otherwise} \end{cases} \tag{3.1}$$

We refer to $M_i\blacktriangle$ as a *positive* indication, to $M_i\blacktriangledown$ as a *negative* indication, and to $M_i \blacktriangleright$ as a *neutral* indication. We define a wildcard, ?, meaning that $M_i?$ can be either $M_i\blacktriangle$, $M_i\blacktriangledown$, or $M_i \blacktriangleright$. In addition, we define the *complement* of an indication as follows: $\overline{M_i\blacktriangle} = M_i\blacktriangledown$, $\overline{M_i\blacktriangledown} = M_i\blacktriangle$, and $\overline{M_i \blacktriangleright} = M_i \blacktriangleright$. We expand the complement to work for sets by taking the complement of each element, and we expand the wildcard to work for sets meaning that any member of the set can have any indication. An *indication set*, $IndSet(C, t, o, d_2, d_3, ..., d_n)$, as shown in Formula 3.2, is a set of all possible combinations of indications (of up to *RuleLen* source measures) that occur for one or more source measures in the same cell. We use *MaxSource* as a threshold for the maximum number of source measures we want to combine in an *IndSet*, and we denote the number of indications in a given *IndSet* by $RuleLen(IndSet)$.

$$\begin{aligned} IndSet(C, t, o, d_2, d_3, ..., d_n) = \{\{&Ind_{i_1}(C, t, o, d_2, d_3, ..., d_n), ..., \\ &Ind_{i_q}(C, t, o, d_2, d_3, ..., d_n), ..., Ind_{i_{RuleLen}}(C, t, o, d_2, d_3, ..., d_n)\} | \\ &1 \leqq RuleLen \leqq MaxSource \ \wedge \ 1 \leqq i_q \leqq p-1\} \end{aligned} \tag{3.2}$$

A sentinel set, $SentSet$, is defined as all indications in a cube, C, given the offset, o, where the source measure indication sets, IS_s, are paired with the indications on the target measure, Ind_p, that occur a given *warning period*, w, later.

$$\begin{aligned} SentSet(C, o, w) = \{&(IS_s, Ind_p(C, t+w, o, d_2, d_3, ..., d_n)) | \\ &IS_s \in IndSet(C, t, o, d_2, d_3, ..., d_n) \wedge \\ &(t, d_2, d_3, ..., d_n) \in C \ \wedge \ (t+w, d_2, d_3, ..., d_n) \in C \\ &\wedge \forall Ind_m \in IS_s : (Ind_m \neq M_m \blacktriangleright)\} \end{aligned} \tag{3.3}$$

We say that $(Ind_{Source}, Ind_{Target}) \in SentSet(C, o, w)$ *supports* the *indication rule* denoted $Ind_{Source} \rightarrow Ind_{Target}$. The *support* of an indication rule, denoted by $IndSupp_{Source \rightarrow Target}$, is the number of $(Ind_{Source}, Ind_{Target}) \in SentSet(C, o, w)$ which support the rule. Similarly, the support of *Source*, $IndSupp_{Source}$, is the number of $(Ind_{Source}, M_p?) \in SentSet(C, o, w)$. In Table 3.1, we have calculated the indications (Formula 3.1) with $\alpha = 10\%$ as well as arranged the indications for the source measures in the largest possible indication set (Formula 3.2) for each month. The combination of the source and target measures is equivalent to a sentinel set

(Formula 3.3) where o and w are both set to 1 month. We see that for example the indication rule $NBlgs\blacktriangle \rightarrow Rev\blacktriangledown$ has a support, $IndSupp_{NBlgs\blacktriangle \rightarrow Rev\blacktriangledown}$, of 2, and the indication $NBlgs\blacktriangle$ has a support, $IndSupp_{NBlgs\blacktriangle}$, of 4.

A generalized sentinel rule is an *unambiguous* relationship between *Source* and *Target*, that consists of one or two indication rules. Therefore, we say that there are only two potential sentinels between a set of source measures, *Source*, and a target measure, *Target*, namely: $Source \rightsquigarrow Target$ or $Source \rightsquigarrow inv(Target)$, where *inv* represents an inverted relationship (intuitively, when source changes up, target changes down and vice versa). The relationships between the two potential generalized sentinel rules and their indication rules are defined in Formula 3.4.

$$\begin{aligned} Source \rightsquigarrow Target &= \{Ind_{Source} \rightarrow Ind_{Target}, \overline{Ind_{Source}} \rightarrow \overline{Ind_{Target}}\} \\ Source \rightsquigarrow inv(Target) &= \{Ind_{Source} \rightarrow \overline{Ind_{Target}}, \overline{Ind_{Source}} \rightarrow Ind_{Target}\} \end{aligned} \tag{3.4}$$

If two contradicting indication rules are both supported in *SentSet*, e.g. $Ind_{Source} \rightarrow Ind_{Target}$ and $Ind_{Source} \rightarrow \overline{Ind_{Target}}$, we use the *contradiction elimination process* (Formula 3.5) to eliminate the indication rules with the least support that have the same premise, but a different consequent, and vice versa. However, in order to reflect the contradiction between the indication rules as a less desired feature, we reduce the support of the "cleansed rule" by deducting the support of the rules we eliminated from the support of the rules we preserve.

$$\begin{aligned} ElimSupp_{Source \rightsquigarrow Target} = IndSupp_{Source \rightarrow Target} &- IndSupp_{Source \rightarrow \overline{Target}} \\ + IndSupp_{\overline{Source} \rightarrow \overline{Target}} &- IndSupp_{\overline{Source} \rightarrow Target} \end{aligned} \tag{3.5}$$

Essentially, we force our generalized sentinel rule to be either $Source \rightsquigarrow Target$ or $Source \rightsquigarrow inv(Target)$, and thereby we effectively eliminate both *contradicting* (same premise but different consequent) and *orthogonal* (different premise but same consequent) indication rules. $ElimSupp$ represents the sum of the support for the indication rule(s) in a sentinel after elimination of its contradictions, and if $ElimSupp_{Source \rightsquigarrow Target}$ is positive, it means that the sentinel $Source \rightsquigarrow Target$ contains the strongest indication rules (as opposed to $Source \rightsquigarrow inv(Target)$). Subsequently, $SentRules(C, o, w)$ (Formula 3.6) conducts the elimination process and extract the generalized sentinel rules from *C* with the offset *o* and the warning period *w*. We note that *SentRules* only contain rules where $ElimSupp > 0$, this way we eliminate sentinels composed by indication rules that completely contradict each other ($ElimSupp = 0$).

$$\begin{aligned} &SentRules(C, o, w) = \\ &\quad \begin{cases} \{Source_s \rightsquigarrow Target_p \mid (Source_s?, Target_p?) \in SentSet(C, o, w)\} \\ \text{if } ElimSupp_{Source_s \rightsquigarrow Target_p} > 0 \\ \{Source_s \rightsquigarrow inv(Target_p) \mid (Source_s?, Target_p?) \in SentSet(C, o, w)\} \\ \text{if } ElimSupp_{Source_s \rightsquigarrow inv(Target_p)} > 0 \end{cases} \end{aligned} \tag{3.6}$$

$$Balance_{Source \rightsquigarrow Target} = \frac{4 \times |A| \times |B|}{(|A| + |B|)^2}$$
$$\text{where } A = IndSupp_{Source \rightarrow Target} - IndSupp_{Source \rightarrow \overline{Target}} \quad (3.7)$$
$$B = IndSupp_{\overline{Source} \rightarrow \overline{Target}} - IndSupp_{\overline{Source} \rightarrow Target}$$

$$SentSupp_{Source \rightsquigarrow Target} =$$
$$\begin{cases} IndSupp_{Source}, \text{if} Balance_{Source \rightsquigarrow Target} = 0 \wedge IndSupp_{Source} > 0 \\ IndSupp_{\overline{Source}}, \text{if} Balance_{Source \rightsquigarrow Target} = 0 \wedge IndSupp_{\overline{Source}} > 0 \\ IndSupp_{Source} + IndSupp_{\overline{Source}}, \text{otherwise} \end{cases} \quad (3.8)$$

$$Conf_{Source \rightsquigarrow Target} = \frac{ElimSupp_{Source \rightsquigarrow Target}}{SentSupp_{Source \rightsquigarrow Target}} \quad (3.9)$$

To determine the quality of a sentinel, $Source \rightsquigarrow Target \in SentRules(C, o, w)$, we define Formulae 3.7 to 3.9. *Balance* (Formula 3.7) is used to determine the degree to which a generalized sentinel rule is uni-directional (*Balance=0*) or completely bi-directional (*Balance=1*), meaning that there are exactly the same amounts of positive and negative indications on the target measure in the data used to discover the rule. *SentSupp* (Formula 3.8) tells us how often the premise of the sentinel occurs, and *Conf* (Formula 3.9) tells us how often, when the premise occurs, the consequent occurs within w time. We denote the minimum threshold for *SentSupp* by σ, the minimum threshold for *Conf* is denoted by γ, and the minimum threshold for *Balance* is denoted by β. With these definitions, we say that a sentinel, $A \rightsquigarrow B$, with an offset, o, and a warning period, w, exists in C when $SentSupp_{A \rightsquigarrow B} \geqq \sigma$, $Conf_{A \rightsquigarrow B} \geqq \gamma$, and $Balance_{A \rightsquigarrow B} \geqq \beta$. We use the following notation when describing a generalized sentinel rule: we use *inv* relative to the first source measure which is never inverted. The order of the source measures in a rule is unimportant for its logic, thus source measures can be ordered as it is seen most fit for presentation purposes. In the case where the rule is uni-directional, we add ▲ or ▼ to both the source and the target measure to express the distinct direction of the sentinel. We add $\wedge$ between the source measures, when there is more than one source measure in a rule, e.g., $A \wedge B \wedge C \rightsquigarrow D$, $A \wedge inv(B) \rightsquigarrow C$, and $A▲ \wedge B▼ \rightsquigarrow C▲$.

If we revert to our running example and apply Formulae 3.4, 3.5, and 3.6 to the data, we get Table 3.2 as output. Using Formulae 3.7, 3.8, and 3.9, we test each rule to see if it meets the thresholds $MaxSource = 3, \sigma = 3, \gamma = 60\%$, and $\beta = 70\%$. The column *Conformance* lists the result of this test. We use same order as Table 3.1 to ease the readability. With regards to the failing rules we should note that the uni-directional rules $NBlgs▲ \rightarrow Rev▼$ and $NBlgs▲ \wedge CstPrb▲ \rightarrow Rev▼$ have higher confidence than the bi-directional rules based on the same indications. However since our threshold for balance is greater than zero, these uni-directional rules would also fail and are thus not listed in the table. In Table 3.2, the generalized sentinel rules found can be ordered by either *RuleLen*, *SentSupp*, *Conf*, *Balance*, or a combination in order to describe the quality of the rules. However, when using an optimization algorithm, e.g, hill climbing, it is desirable to be able to describe the quality of a rule

with just a single number. For this purpose we denote the maximal value of *ElimSupp* for any sentinel in $SentRules(C, o, w)$ by $MaxElimSupp(C, o, w)$.

$$\begin{aligned} Score(Source \rightsquigarrow Target) &= (1 - wp + \frac{(1 + Maxw - w) \times wp}{Maxw}) \\ &\times (\frac{1}{2} + \frac{1 + MaxSource - RuleLen(Source)}{MaxSource \times 2}) \times \frac{ElimSupp_{Source \rightsquigarrow Target}}{MaxElimSupp(C, o, w)} \\ &\times Conf_{Source \rightsquigarrow Target} \times (\frac{1}{2} + \frac{Balance_{Source \rightsquigarrow Target}}{2}) \end{aligned} \tag{3.10}$$

We define *Score* for a sentinel, $Source \rightsquigarrow Target \in SentRules(C, o, w)$, as shown in Formula 3.10. With this definition of *Score*, we introduce the threshold, *Maxw*, which is the maximum length of the warning period, *w*, we are willing to accept. The constant, *wp*, represents the warning penalty, i.e., the degree to which we want to penalize rules with a higher *w* (0=no penalty, 1=full penalty). The idea of penalizing higher values of *w* is relevant if a pattern is cyclic, e.g., if the indication of a sentinel occurs every 12 months, and the relationship between the indications on the source measure(s) and the target measure is less than 12 months, then a given rule with a warning period *w* is more desirable than the same rule with a warning period *w*+12. We also take into consideration that it is desirable to have shorter, general rules with low *RuleLen*. This prevents our algorithm from "overfitting" rules [45] and thus generating very specific and therefore irrelevant rules. In addition, *Score* takes into consideration the actual number of times the rule occurs in a cube, adjusted for contradictions, *ElimSupp*, as well as the confidence, *Conf*, of the rule. Finally, we consider the *Balance* of the rule, since we have a preference for rules that are bi-directional. In Table 3.2 the ordering by *Score* has proven useful, and we note that the two bottom rules with the lowest *Score* are also the rules that fail to meet the thresholds we set. Given these thresholds, and constants set to $wp = \frac{1}{2}, Maxw = 10$, we would expect a conforming rule to have: $Score \geqq (1 - \frac{1}{2} + \frac{(1+10-1)\times\frac{1}{2}}{10}) \times (\frac{1}{2} + \frac{1+3-3}{3\times2}) \times \frac{3\times0.6}{4} \times 0.6 \times (\frac{1}{2} + \frac{0.7}{2}) = 0.15$. We should note that this is only a "rule of thumb" since the values in Formula 3.10 may

SentRules	*RuleLen*	*SentSupp* (*ElimSupp*)	*Conf*	*Balance*	*Score*	*OK* ?
$CstPrb \wedge inv(WHts) \rightsquigarrow inv(Rev)$	2	4 (4)	100%	100%	0.83	OK
$WHts \rightsquigarrow Rev$	1	6 (4)	67%	100%	0.67	OK
$NBlgs \wedge CstPrb \wedge inv(WHts) \rightsquigarrow inv(Rev)$	3	3 (3)	100%	89%	0.47	OK
$CstPrb \rightsquigarrow inv(Rev)$	1	5 (3)	60%	89%	0.43	OK
$NBlgs \wedge inv(WHts) \rightsquigarrow inv(Rev)$	2	5 (3)	60%	89%	0.35	OK
$NBlgs \wedge CstPrb \rightsquigarrow inv(Rev)$	2	4 (2)	50%	0%	0.10	Failed
$NBlgs \rightsquigarrow inv(Rev)$	1	6 (2)	33%	0%	0.08	Failed

Table 3.2: Sentinels ordered by their respective *Conformance* and *Score*.

vary, thus the thresholds needs to be inspected individually to determine if a rule is conforming or not. With *Score* as a uniform way to assess the quality of a generalized sentinel rule, we can now define *Optimalw*(C,o), as shown in Formula 3.11, which is the value of *w*, $1 \leqq w \leqq Maxw$, where *SentRules*(C,o,w) contains the rule with the highest *Score*. The reason for the construction details of *Score* is elaborated further in Chapter 5.

$$\begin{aligned} &Optimalw(C,o) = w \text{ such that } 1 \leqq w, w' \leqq Maxw \ \wedge \ \exists S \in SentRules(C,o,w): \\ &\qquad (\forall w' \neq w : (\forall S' \in SentRules(C,o,w') : (Score(S) \geqq Score(S')))) \end{aligned} \tag{3.11}$$

$$\begin{aligned} &SentRulesPruned(C,o,w) = S \in SentRules(C,o,w) \mid \\ &\nexists S' \in SentRules(C,o,w): \ (Score(S') \geqq Score(S) \ \wedge \ Ind_{Source_{S'}} \subset Ind_{Source_S})\} \end{aligned} \tag{3.12}$$

Having found the optimal *w*, it is also desirable to prune the generalized sentinel rules such that we only output the best rules in terms of *Score*, and the shortest rules in terms of number of source measures. For this purpose, we use the *SentRulesPruned* function, as shown in Formula 3.12, that eliminates rules with poor quality (lower *Score*) if a shorter rule exists with at least as good a *Score*, and where the indication set is a proper subset of the longer rule.

We say that $SentRulesPruned(C, o, Optimalw(C, o))$ ordered by their respective *Score* are the best sentinels in a database, *C*, with the offset, *o*. Using the *SentRulesPruned* function, we note that $NBlgs \wedge CstPrb \wedge inv(WHts) \rightsquigarrow inv(Rev)$ in third line in Table 3.2 would be eliminated since the shorter rule $CstPrb \wedge inv(WHts) \rightsquigarrow inv(Rev)$ has a better *Score*. In other words, we do not improve the quality by adding *NBlgs*.

3.3 Discovering Generalized Sentinel Rules

Preliminaries: To discover all generalized sentinel rules in a cube, *C*, we intuitively need to test all possible rule combinations where the number of source measures combined varies from 1 to *MaxSource*, and the warning period, *w*, varies from 1 to *Maxw*. However, as an alternative to this brute force approach, we apply two good approximations to improve performance, namely, *Reduced Pattern Growth* (RPG) to optimize the combining of source measures, and *hill climbing* to optimize the auto-fit of warning periods. Intuitively, it is not hard to imagine that the number of source measures has a significant impact on the performance of the algorithm since they can each have indications in two directions, and all of these directions can be combined. This means that the total number of combinations for *k* source measures is $\sum_{x=1}^{l} \frac{2k!}{(2k-x)!}$ where $l = MaxSource$. If we preserve the order of the source measures we can reduce the number of potential rules (permutations) to $\sum_{x=1}^{l} \frac{2k!}{l!(2k-x)!}$, however, the

number of combinations still explode when a significant amount of source measures needs to be examined. Therefore, there is a performance reward if we can *prune* the source measures that are unlikely to become part of any good rule, at an early stage in the algorithm. From experiments on real-world data, we know that the likelihood of individual source measures being part of a good rule can be described as a *power law*, meaning that a few source measures are very likely to be part of many good rules for a given target measure, whereas the majority of source measures are not likely to be part of a good rule at all. Given this property of source measures, the ability to prune has a significant potential for performance improvement. In addition to reducing the number of source measures we examine, we can save time and memory by storing only the best rules in memory. The RPG optimization, described below, has these two abilities.

The Table of Combinations (TC) (Table 3.3) is an intermediate table used for optimization. The TC is generated in one pass over the cube, *C*. We use a sliding window of $Maxw + o$ rows in memory for each combination of $(d_2, d_3, ..., d_n)$ to update the indications (Formula 3.1) on source and target measures for $w \in \{1, 2, ..., Maxw\}$. For each occurrence of combined source measure indications, $(Ind_{m_1}, ..., Ind_{m_{p-1}})$, the target measure indication, Ind_{m_p}, is calculated for all values of $w \in \{1, 2, ..., Maxw\}$, and the indications, $ElimSupp_w$, are mapped to integers as follows: Inc→1, Dec→-1, and Neutral→0. The discretized values are added to the fields $ElimSupp_w$ on the unique row in TC for the combination $(Ind_{m_1}, ..., Ind_{m_{p-1}})$. In addition, the field *CombSupp* on this row is increased by 1. If a combination, $(Ind_{m_1}, ..., Ind_{m_{p-1}})$, does not exist in TC, an additional row is appended with the new combination. The TC holds all indication rules with $RuleLen(Source) = p - 1$ with their respective *ElimSupp* (Formula 3.5), denoted by $ElimSupp_{w=x}$ for all $x \in \{1, 2, ..., Maxw\}$. We store the additional information about the direction of the target measure in the sign of $ElimSupp_w$ (positive=up,

Ind_{M_1}	Ind_{M_2}	Ind_{M_3}	...	$Ind_{M_{p-1}}$	*CombSupp*	*ElimSupp* $w=1$	*ElimSupp* $w=2$	*ElimSupp* $w=3$	... ...	*ElimSupp* $w=Maxw$
Neutral	Neutral	Dec		Dec	1	1	-1	-1		1
Neutral	Inc	Inc		Inc	2	0	-2	1		1
Neutral	Dec	Dec		Dec	3	-1	3	1		-1
Neutral	Dec	Neutral		Dec	1	1	1	-1		0
Neutral	Dec	Dec		Inc	1	-1	0	0		-1
Neutral	Neutral	Inc		Dec	1	0	0	-1		1
Neutral	Inc	Neutral		Dec	1	0	-1	1		1
Neutral	Inc	Inc		Inc	1	-1	1	1		-1
Inc	Inc	Dec		Dec	4	3	4	-1		0
Dec	Inc	Inc		Inc	3	-3	-3	0		0

Table 3.3: Logical Table of Combinations (TC) Example

negative=down). The field *CombSupp* in TC holds the support of the given source measure indication combination, and is used for calculating *SentSupp* (Formula 3.8). Once generated, the TC is a highly compressed form representing the information needed to mine all potential sentinel rules in *C*. In the TC, a rule, $SentRule \in SentRules(C, o, w)$, has $SentSupp = \sum CombSupp$ (Formula 3.8) when selecting the rows that support either $Ind_{Source_{SentRule}}$ or $\overline{Ind_{Source_{SentRule}}}$. Similarly, the components for *Balance* (Formula 3.7) can be found as $A = \sum_{x \in X} ElimSupp_w$ where *X* are the rows that support $Ind_{Source_{SentRule}}$, and $B = \sum_{y \in Y} ElimSupp_w$ where *Y* are the rows that support $\overline{Ind_{Source_{SentRule}}}$. We recall from Formulae 3.4 and 3.6 that a bi-directional sentinel has $Balance > 0$ and thus require a pair of indication rules with opposite directions.

In Table 3.3, the sentinel $M_1 \rightsquigarrow M_p$ has $SentSupp_{M_1 \rightsquigarrow M_p} = 7$, $A = 3$, and $B = -3$, when $w = 1$ (as seen in column $ElimSupp_{w=1}$), meaning that the rule has $ElimSupp_{M_1 \rightsquigarrow M_p} = |A| + |B| = 6$. We note that A and B have been cleansed for contradictions prior to insertion in the TC, and that the sign is a property used by the TC and should thus be omitted. We have $Conf_{M_1 \rightsquigarrow M_p} = 0.857$ and $Balance_{M_1 \rightsquigarrow M_p} = 1$ when the warning period, *w*, is 1. Similarly, we can find the values for $M_3 \wedge M_{p-1} \rightsquigarrow inv(M_p)$ by inspecting the rows where $M_3 \neq$ Neutral $\wedge$ $M_{p-1} \neq$ Neutral. If we set $w = 2$, we find $SentSupp_{M_3 \wedge M_{p-1} \rightsquigarrow inv(M_p)} = 14$, $A = -4$, and $B = 6$, $ElimSupp_{M_3 \wedge M_{p-1} \rightsquigarrow inv(M_p)} = 10$, thus $Conf_{M_3 \wedge M_{p-1} \rightsquigarrow inv(M_p)} = 0.714$. In addition, we have $Balance_{M_3 \wedge M_{p-1} \rightsquigarrow inv(M_p)} = 0.960$. A sentinel is therefore typically combined from multiple rows in the TC, i.e., a rule with $RuleLen(Source) \leqq MaxSource$ will need a full scan of TC to identify *ElimSupp*, *Balance*, and *SentSupp* because $MaxSource << p$. Since we do not know which source measure indications occur at the same time, there is no generic sorting method that can optimize the scans further.

Reduced Pattern Growth (RPG) is a method that delivers a good approximation of the top sentinel rules, and which is much more efficient than a full pattern growth of all combinations of source measures. The quality of the approximation is examined in detail in Section 3.4, specifically in Figure 3.4(a). In the RPG process, we identify the source measures, that are *most likely* to be part of the best generalized sentinel rules for a given value of *w*, as an alternative to testing *all* combinations of source measures. We do this by inspecting the *influence* of the source measure in the TC, defined as the number of rows in the TC in which a source measure has indications different from neutral while at the same time the indication on the target measure is different from neutral (zero in Table 3.3). With this definition we can assess the influence, *Inf*, of all source measures for each value of *w* as shown in Table 3.4. We note that the source measure M_{p-1} is the most influential from our TC (Table 3.3),

w	Inf_{M_1}	Inf_{M_2}	Inf_{M_3}	$Inf_{M_{p-1}}$	Inf_{All}	$Inf_{M_2+M_3+M_{p-1}}$	$Pareto$
1	2	6	6	7	22	19	86%
2	2	7	6	8	25	21	84%
3	1	6	6	8	24	20	83%
...							
Maxw	0	5	6	7	18	18	100%

Table 3.4: Source Measure Influence & Pareto Example

specifically it has an influence of 8 for values $w = 2$ and $w = 3$, and it has an influence of 7 for values $w = 1$ and $w = Maxw$. With the notion of the source measures "behaving" in accordance with a power law, we apply a Pareto principle to select only the most influential source measures, meaning that we select the source measures with a total influence that account for more than $RPGpareto$ % of the sum of the influence of all source measures. In Table 3.4 we see that source measures M_2, M_3, and M_{p-1} account for more than 80% of the influence for all values of *w*, i.e. setting $RPGpareto = 80\%$ would mean that we only consider these three measures for the values of *w* shown in the table. Alternatively, setting $RPGpareto = 85\%$ would mean that source measure M_1 would also be included in the pattern growth for values $w = 2$ and $w = 3$. From this point, we *grow* sentinels from the measures identified. Starting with 1 source measure, we add the remaining influential source measures one at a time to create longer rules until the maximum number of source measures we desire is reached. In this process we only store a sentinel, and continue to add source measures, if the added source measures give a higher *Score*.

Hill Climbing identifies the warning period, *w*, where the sentinel with the highest *Score* exists as an alternative to calculating all max(*Score*) for all *w*. We optimize the hill climbing process by changing *w* +22 in the direction of the local maximum while *Score* increased. Once *Score* decreases, we have passed a local maximum, and we test *Score* for $w - 1$ as well to ensure that we have not stepped over the local maximum. During the hill climb, the set of sentinels with the highest *Score* resides in memory until a better set for another value of *w* is found. Upon termination, the best set of generalized sentinel rules $SentRulesPruned(C, o, Optimalw(C, o))$ resides in memory.

The SentHiRPG Algorithm: We assume without loss of generality that of the p measures in the cube, *C*, $M_1 ... M_{p-1}$ are the source measures and M_p is the target measure.

Step 1 scans the cube, *C*, and builds the *Table of Combinations (TC)* (Table 3.3). Since the data is received in sorted order by the time-dimension, *t*, for each combination of $(d_2, d_3, ..., d_n)$, we only need a sliding window of $Maxw + o$ rows in memory to update the indications (Formula 3.1) on source and target measures for $w \in \{1, 2, ..., Maxw\}$. Using the procedure *UpdateTC*, each unique combination of source measure indications, $(Ind_{m_1}, ..., Ind_{m_{p-1}})$, that exists in the cube, *C*, is added

Algorithm: SentHiRPG
Input: A list of facts from a cube, *C*, ordered by $(d_2, d_3, ..., d_n, t)$, an offset, *o*, a maximum warning period length, *Maxw*, a maximum number of source measures per rule, *MaxSource*, a warning penalty, *wp*, a threshold for RPG, *RPGpareto*, a threshold for indications, α, a minimum *SentSupp* threshold, σ, a minimum *Conf* threshold, γ, and a minimum *Balance* threshold, β.
Output: Sentinel rules with a given warning period, *Optimalw*, and their respective *SentSupp*, *Conf*, *Balance*, and *Score*.
Method: Sentinel rules are discovered as follows:

Procedure *UpdateTC*. For each cell pair, $\{(t, d_2, d_3, ..., d_n), (t + o, d_2, d_3, ..., d_n)\}$ in memory, calculate the indications (Formula 3.1) using α on source measures $m_1 ... m_{p-1}$, discretize $Ind_{m_i} \in (Ind_{m_1}, ..., Ind_{m_{p-1}})$ as Inc, Dec, or Neutral. If combination $(Ind_{m_1}, ..., Ind_{m_{p-1}})$ does not already exist in *Table of Combinations (TC)*, append row to TC. For each $w \in \{1, 2, ..., Maxw\}$, for the combination $(Ind_{m_1}, ..., Ind_{m_{p-1}})$, update the value of the indication, $ElimSupp_w$, by adding the indication (Formula 3.1) based on the pairs, $\{(t + w, d_2, d_3, ..., d_n), (t + w + o, d_2, d_3, ..., d_n)\}$ in memory, discretized as 1, -1, or 0. In addition, increase the value of the combination support counter, *CombSupp*, by 1.

Function *RPGmeasures(w)*; Returns a set of source measures. For each source measure, $M_i \in \{M_1 ... M_{p-1}\}$, calculate *influence* as $\sum ElimSupp_w$ for all rows in TC where $m_i \neq$ Neutral. Return source measures in ascending order of *influence* until $\frac{\sum influence \text{ of source measures returned}}{\sum influence \text{ of all source measures}} \geqq RPGpareto$.

1. Scan C, and when the sliding window of $Maxw + o$ rows are in memory perform *UpdateTC* whenever a new row is read. From this point keep only $Maxw + o$ rows in memory by disregarding the oldest row whenever a new row is read until a new combination of $(d_2, d_3, ..., d_n)$ occurs, at this point flush all rows and load new $Maxw + o$ rows for the next combination of $(d_2, d_3, ..., d_n)$. Repeat *UpdateTC* whenever a new row is read until the scan of C is complete, and while the sliding window of $Maxw + o$ rows exist in memory.
2. Find the value of *w* that corresponds to *Optimalw(C,o)* (Formula 3.11) by *hill climbing* on the value of *max(Score(SentRules(C,o,w)))* (Formulae 3.6 and 3.10). The generalized sentinel rules for each tested *w*, *SentRules(C,o,w)*, are "grown" from 1 to *MaxSource* in *RuleLen* by combining the source measures returned by *RPGmeasures(w)*. Only rules with source measure combinations where *Score* improves when adding an additional source measure are stored, meaning that the output will be equivalent to $SentRulesPruned(C, o, w)$ (Formula 3.12). While testing different values of *w*, the sentinels with the highest score at any given time is stored in memory and not flushed until a better set of rules exists for another value of *w*.
3. Output the "best" generalized sentinel rules from memory, i.e., $SentRulesPruned(C, o, Optimalw(C, o))$ (Formula 3.12), where $SentSupp >= \sigma$, $Conf => \gamma$, and $Balance >= \beta$.

or updated in the TC. For each source measure combination, the indications on the target measure, Ind_{m_p}, are calculated for all values of $w \in \{1, 2, ..., Maxw\}$, and the indications, $ElimSupp_w$, are mapped to integers as follows: Inc→1, Dec→-1, and Neutral→0. The discretized values are added to the fields $ElimSupp_w$ on the unique row in TC for the combination $(Ind_{m_1}, ..., Ind_{m_{p-1}})$. In addition, the field *CombSupp* on the row is increased by 1 each time the source measure combination occurs. In other words, as we scan the cube, *C*, new unique combinations of $(Ind_{m_1} ... Ind_{m_{p-1}})$ make the number of rows in the TC grow. Every time one of these combinations are found during the scan, 1, -1, or 0 is added to the corresponding value for $ElimSupp_w$, and the value of *CombSupp* is increased by 1. Following Step 1, we have the contribution to *ElimSupp* (Formula 3.5) for all values of $w \in \{1, 2, ..., Maxw\}$ as well as SentSupp (Formula 3.8) for each source measure combination in the TC.

When selecting the rows that support either $Ind_{Source_{SentRule}}$ or $\overline{Ind_{Source_{SentRule}}}$ in the TC, a rule, $SentRule \in SentRules(C, o, w)$, has $SentSupp = \sum CombSupp$ (Formula 3.8). Similarly, the components for *Balance* (Formula 3.7) can be found as $A = \sum_{x \in X} ElimSupp_w$ where *X* are the rows that support $Ind_{Source_{SentRule}}$, and $B = \sum_{y \in Y} ElimSupp_w$ where *Y* are the rows that support $\overline{Ind_{Source_{SentRule}}}$. We should note that since we express the direction (Inc or Dec) with the sign of the indications on the target measure, $ElimSupp_w$, we need to use the *absolute values* for each direction when calculating *ElimSupp*, thus we have $ElimSupp = |A| + |B|$ (Formulae 3.5 and 3.7). In the conceptual description of the TC, we calculated $SentSupp_{M_1 \rightsquigarrow M_p} = 7$, $Conf_{M_1 \rightsquigarrow M_p} = 0.857$ and $Balance_{M_1 \rightsquigarrow M_p} = 1$ for the sentinel $M_1 \rightsquigarrow M_p$ when $w = 1$. Calculating *Score* with these values allows us to compare the quality of any sentinel combined from the source measures and the target measure in *C* using the TC.

In Step 2 we identify the best value of *w*, *Optimalw(C,o)* (Formula 3.11), which is defined as the value of *w* where the sentinel(s) with the highest *Score* exist(s) (Formula 3.10). We use two optimization techniques for this purpose: hill climbing and Reduced Pattern Growth as explained above. Hill climbing is a well-known optimization technique [33] and it is an alternative to testing all values of $w \in \{1, 2, ..., Maxw\}$ to identify *max(Score(SentRules(C,o,w)))* (Formulae 3.6 and 3.10). During the hill climbing process, whenever a value of *w* needs to be inspected to identify *max(Score(SentRules(C,o,w)))*, we apply the Reduced Pattern Growth (RPG) function, *RPGmeasures*. Having identified the most influential source measures, we "grow" the sentinels with *RuleLen=1* to "longer" rules until *RuleLen=MaxSource* by combining the source measures returned by *RPGmeasures* for a given value of *w*. During this process we only store sentinels with greater *RuleLen* if the extended *RuleLen* translates into a higher *Score*. This means that we are growing a set of sentinels

equivalent to $SentRulesPruned(C, o, w)$ (Formula 3.12). Once all sentinels have been grown for a particular value of *w*, the *max(Score(SentRules(C,o,w)))* value is returned to the hill climbing process to determine whether to examine more values of *w* or not. During the entire hill climb, the set of sentinels with the highest *Score* so far is stored in memory until a better set is found for another value of w. Upon termination, the best set of generalized sentinel rules $SentRulesPruned(C, o, Optimalw(C, o))$ resides in memory.

In Step 3 the sentinels that conform with the thresholds for *SentSupp*, *Conf* and *Balance* from the set $SentRulesPruned(C, o, Optimalw(C, o))$ are output from memory.

Computational Complexity: The algorithm has a complexity of $\mathcal{O}(n+c\times p(q)^l \times k^l \times m)$ where *n* is the size of *C*, *c* is the size of TC, *p* the percentage of remaining source measures expressed as a function of *q*, where *q* is *RPGpareto*, *l* is *MaxSource*, and *m* is *Maxw*. In Section 3.4 we verify this assessment through extensive experiments.

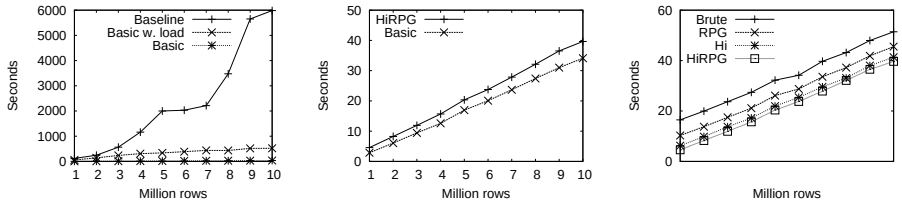
Implementation: The SentHiRPG algorithm variants were implemented in Microsoft C# and compiled into a stand-alone 64-bit executable file. The initial version loaded the data directly from a Microsoft SQL Server during Step 1. However, this approach was not able to feed the data fast enough to stress test the algorithm. As a consequence, we loaded all data into main memory and from here into Step 1. This approach is realistic (see Section 3.4) and it provided sufficient bandwidth to stress the algorithm in order to see the effect of the optimizations applied. The TC built in Step 1 is stored in a hash table where each source measure indication for a row is encoded into 2 bits. In Step 2, when testing a given value of *w*, we found that testing all combinations of source measures from $RuleLen = 1$ to $RuleLen = MaxSource$, and storing only longer rules if *Score* improved, was far more efficient than a genetic algorithm without mutation. We use the following algorithm variants: **Brute**: brute force, both optimization options are off. **Hi**: **Hi**ll climbing optimization activated, RPG off. **RPG**: **R**educed **P**attern **G**rowth activated, hill climb off. **HiRPG**: both **Hi**ll climb & **R**educed **P**attern **G**rowth activated.

3.4 Experiments

We use a range of synthetic datasets and a range of real-world datasets for the experiments. The synthetic datasets closely resemble our running example and have

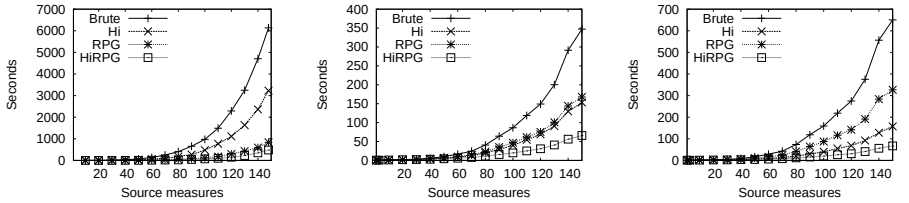
the same rule relationships, since the three source measures are duplicated to create a dataset with any number of source measures. The synthetic datasets range from 1,000,000 to 10,000,000 rows in 1,000,000 row intervals, with 50 source measures and one target measure. We note that the sizes of these datasets are *huge* compared to the real-world dataset. In general, we would expect any real application of sentinels to work on *significantly fewer rows* since we typically aggregate the data, e.g., into months or weeks, before finding sentinels. In addition, we have generated datasets with 1,000,000 rows and with 1, 10, 20, 50, 100 and 150 source measures and one target measure. The real-world datasets are produced from the operational data warehouse of TARGIT A/S. Based on experience with more than 3,800 customers worldwide, we will characterize this dataset as typical for a medium-sized company with a mature data warehouse. The original dataset contains 241 months (20.1 years) of operational data scattered across 148 source measures. Descendants of this dataset are produced by selecting a given number of source measures randomly to produce datasets with 10, 20, 30, ..., 140 source measures. When nothing else is specified, the synthetic dataset has 1,000,000 rows, and the algorithm has the following settings: *wp*=0.5, *MaxSource*=3, *Pareto*=85%, and thresholds: *SentSupp*=3, *Conf*=0.6, and *Balance*=0.7.

Scaling rows: In Figure 3.1 we validate that "HiRPG" scales linearly to 10 million rows of data. In Figure 3.1(a) a "simple Brute" with TC optimization alone was far more efficient than the baseline from prior art (Chapter 2). In Figure 3.1(b) we compare "simple Brute" with "HiRPG "; the distance between the lines is the cost of auto-fitting *w* over 50 periods and combining 50 source measures. As expected, based on our assessment of computational complexity, we observe "simple Brute" and "HiRPG" to scale linearly in the number of rows (when the other factors are kept constant). We observe the difference between "simple Brute" and "HiRPG" to be close to constant for an explanation of the slight increase in runtime. In Figure 3.1(c) the variants scale linearly as expected, and not surprisingly the fully optimized "HiRPG" is best.



(a) Baseline comparison (b) Cost of hill climbing & RPG (c) Scalability of algorithm variants

Figure 3.1: Scaling the number of rows

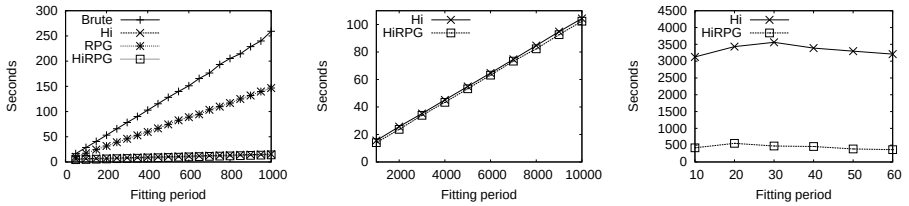


(a) Real data, $Maxw = 24$ (b) Synthetic data, $Maxw = 24$ (c) Synthetic data, $Maxw = 50$

Figure 3.2: Scaling the number of source measures

Scaling source measures: Scaling the number of source measures on real data has an exponential impact on all variants, but "HiRPG" and "RPG" are very efficient in reducing this impact. On the comparable synthetic dataset in Figure 3.2(b) "RPG"and "Hi" are almost equal in efficiency, and we see "Hi" excelling when expanding the fitting period in Figure 3.2(c). We attribute this to the existence of a true power law in the real data, whereas in the synthetic data the relationships between source and target measures are simply repeated for every three measures which means that many measures have strong relationships. The fact that "Hi" improves further when increasing *Maxw* is not surprising since hill climbing specifically reduces the cost of increasing *Maxw*. We note that for the synthetic data "HiRPG" is still by far the most efficient variant of the algorithm, and although the dominant computational complexity is cubic in the number of source measures ($MaxSource = 3$ and the other factors are also kept constant), the RPG optimization significantly reduces this impact.

Scaling the fitting period: In this experiment, we vary the *Maxw* over which we fit the warning period, *w*. In Figure 3.3(a) and (b) we see that "HiRPG" is performing best when scaling *Maxw*, followed by "Hi", that lack the RPG optimization. Both variants scale linearly to the extreme fitting over 10,000 periods. In Figure 3.3(c) the same lack of RPG optimization is more evident on real data, given the power law

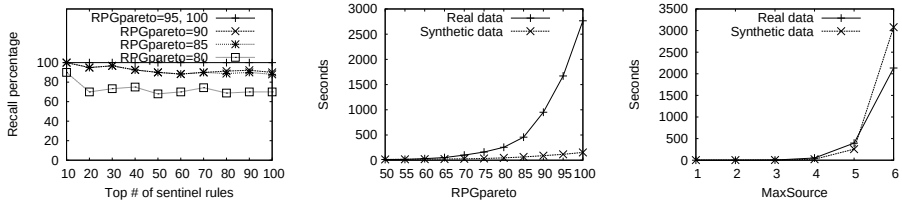


(a) Synthetic data (b) Synthetic data, best variants (c) Real data

Figure 3.3: Scaling the fitting of warning period

explained earlier. The slight decrease in runtime on the higher values of *Maxw* should be seen in the context that the dataset has only 241 rows. Therefore, we should not interpret the findings on the real dataset as sub-linear scalability. In other words, we note that scaling the fitting of *w* scales linearly as expected from our assessment of computational complexity (when the other factors are kept constant).

Scaling parameters *RPGpareto* and *MaxSource*: In Figure 3.4(a) we see the recall ($\frac{\text{Number of Sentinels - false negatives}}{\text{Number of Sentinels}}$) of the top 10 to 100 top-*Score* sentinels for "HiRPG". We note the significant drop in the recall at $RPGpareto = 80$. In Figure 3.4(b) we see performance for "HiRPG" when scaling *RPGpareto*. We notice the impact cost when $RPGpareto > 85$. Combining Figure 3.4(a) and (b) suggests a recall "sweet-spot" at $RPGpareto = 85$ (100% of top 10, and 88% of top 100) before the performance cost "explodes". In Figure 3.4(c) we scale *MaxSource* for "HiRPG". We note that performance on the real dataset passes the synthetic dataset as the complexity "explodes". We attribute this shift to the power law in the real dataset, i.e., as *MaxSource* increases, so does the effect of the RPG process.



(a) Recall on Real Data, $Maxw = 24$ (b) Scaling *RPGpareto* (c) Scaling *MaxSource*

Figure 3.4: HiRPG Performance when scaling parameters *RPGpareto* and *MaxSource*

Qualitative Experiment: Apart from assessing the performance of SentHiRPG, we also found interesting and business relevant sentinels on the real-world data, e.g, IF the number of people involved in a customer decision process decrease AND the revenue from training increase, both by 10% or more, THEN the total revenue for TARGIT A/S is expected to increase by 10% or more within three months; and vice versa. In this particular case for TARGIT A/S, it was surprising that the number of people involved in the decision process could be used as an indicator, whereas it has been known for some time that selling more training will typically make a customer expand his solution. Intuitively, it makes sense that if more people are involved in a decision process, then it takes more time, and therefore less revenue will be generated on the short-term. In Chapter 5 these sentinels and their business potential is described in greater detail.

3.5 Conclusion and Future Work

We have proposed a novel approach for discovering so-called generalized sentinel rules (sentinels) in a multi-dimensional data cube for business intelligence. We extended prior work to allow multiple measures to be combined into better sentinels using the novel qualitative measures *Balance* and *Score*. In addition, these measures were used to auto-fit the best warning period for the sentinel. We presented an algorithm that, given a target measure, could autonomously find the best warning period and output the best sentinels from this. In the algorithm we introduced a novel table of combinations (TC) and a reduced pattern growth (RPG) approach, and we demonstrated the optimization effect of these approaches in combination with a hill-climbing optimization to produce from ten to twenty times improvement in performance. We showed that our optimized algorithm scales linearly on large volumes of data and when fitting warning period over large period intervals, and that it scales close to linearly when combining large sets of source measures. We have previously demonstrated that sentinels can find strong and general rules that would not be found by sequential pattern mining or correlation techniques (Chapter 2), this obviously holds even more for generalized sentinel rules.

For future work, a natural development would be to mine sentinels for multiple target measures simultaneously to improve performance. Secondly, we could exploit the multi-dimensional environment by having sentinel mining fit the aggregation level on dimensions as well as select the location and shape of the data area. Third, a parallelization of SentHiRPG could improve scaling to datasets with even more source measures.

Chapter 4

Using Sentinel Technology in the TARGIT BI Suite

This chapter demonstrates so-called *sentinels* in the TARGIT BI Suite. Sentinels are a novel type of rules that can warn a user if one or more *measure changes* in a multi-dimensional data cube are expected to cause a change to another measure critical to the user. We present the concept of sentinels, and we explain how sentinels represent *stronger* and more *specific* rules than sequential patterns and correlation techniques. In addition, we present the algorithm, implementation, and data warehouse setup that are prerequisites for our demo. In the demo we present a dialogue where users, without any prior technical knowledge, are able to select a critical measure, a number of cubes, and a time dimension, and subsequently mine and schedule sentinels for *early warnings*.

4.1 Introduction

Bringing data mining to the masses has been a failed quest by major business intelligence (BI) vendors since the late 1990s [50]. From a business perspective, there is an obvious potential to use the available computing resources to allow users in an OLAP environment to use data mining for exploring data relationships that are practically impossible to find manually. We believe that integration of BI disciplines and usability is the key to unlock the big potential of end user data mining that has not yet reached the business users. With this in mind, we have implemented so-called *sentinels* in the TARGIT BI Suite that is currently available to more than 274,000 users across more than 3,800 organizations world-wide.

A sentinel is a novel type of causal rule-based relationship; the concept and formal definitions have been developed in a collaborative research project between TARGIT A/S and Aalborg University (Chapters 2 and 5). Sentinels are discovered through a data mining process, where changes in one or multiple *source measures* are followed by changes to a *target measure* (typically a KPI), within a given time period, referred to as the *warning period*. An example of a sentinel for a company could be: "IF Number of Customer Problems go up and Website Traffic Volume goes down THEN Revenue goes down within two months AND IF Number of Customer Problems go down and Website Traffic Volume goes up THEN Revenue goes up within two months". Such a rule will allow a BI system to notify a user to take corrective action, e.g., if "Number of Customer Problems go up and Website Traffic Volume goes down".

Sentinels are based on the Computer Aided Leadership & Management (CALM) theory [36]. The idea in CALM is to take the Observation-Orientation-Decision-Action (OODA) loop (originally pioneered by "Top Gun" fighter pilot John Boyd in the 1950s), and integrate BI technologies to drastically increase the speed with which a user "travels" through the OODA loop. Sentinels improve the speed of the OODA loop's observation and orientation phases by giving the decision maker an early warning (faster observation) that threatens a KPI. At the same time, the sentinel highlights the threat (faster orientation) by listing the measure changes that appears to be "causing" it. In other words, sentinels contribute with both synergy and efficiency for a user cycling an OODA loop.

Sentinels are mined on the measures and dimensions of multiple cubes in an OLAP database, as opposed to the "flat file" formats used by most traditional data mining methods. Sentinels find rules that would be impossible to detect using traditional techniques, since sentinels operate on *data changes* at the *schema level* as opposed to absolute *data values* at the *data level* such as association rules and sequential patterns typically do [5]. As explained in Section 4.2, the *bi-directional* sentinels are *stronger* rules than those mined by sequential pattern mining (Chapter 5). In addition, sentinels are more *specific* than the relationships that can be found using regression techniques such as [60]. In this context, sentinels are a set of *"micro-predictions"* that are complementary to regression and correlation techniques. Sentinels are useful for discovering strong relationships between a smaller subset within a dataset (Chapter 2), and thus they are useful for detecting warnings whenever changes (that would otherwise go unnoticed) in a relevant source measure occur.

In Section 4.2, we present the concept of sentinels with a data example. In Sections 4.3 and 4.4 we present the sentinel mining algorithm and its context in the TARGIT BI Suite. In Section 4.5 we demonstrate sentinel mining from a user context in the TARGIT BI Suite. In Section 4.6 we conclude and present market feedback.

(a) Source

Month	*PeoInv*	*UniRev*	Change
2008-Jan	1	115	
2008-Feb	2	115	$PeoInv$▲
2008-Mar	2	100	$UniRev$▼
2008-Apr	3	90	$PeoInv$▲,$UniRev$▼
2008-May	2	363	$PeoInv$▼,$UniRev$▲
2008-Jun	3	310	$PeoInv$▲,$UniRev$▼
2008-Jul	2	440	$PeoInv$▼,$UniRev$▲
2008-Aug	4	297	$PeoInv$▲,$UniRev$▼
2008-Sep	5	260	$PeoInv$▲,$UniRev$▼
2008-Oct	6	230	$PeoInv$▲,$UniRev$▼
2008-Nov	4	294	$PeoInv$▼,$UniRev$▲
2008-Dec	5	264	$PeoInv$▲,$UniRev$▼
2009-Jan	6	230	$PeoInv$▲,$UniRev$▼
2009-Feb	4	270	$PeoInv$▼,$UniRev$▲
2009-Mar	3	353	$PeoInv$▼,$UniRev$▲
2009-Apr	2	400	$PeoInv$▼,$UniRev$▲

(b) Target

Month	*Rev*	Change
2008-Apr	900	
2008-May	1001	Rev▲
2008-Jun	1200	Rev▲
2008-Jul	750	Rev▼
2008-Aug	1001	Rev▲
2008-Sep	1100	
2008-Oct	1250	Rev▲
2008-Nov	970	Rev▼
2008-Dec	850	Rev▼
2009-Jan	720	Rev▼
2009-Feb	1250	Rev▲
2009-Mar	930	Rev▼
2009-Apr	800	Rev▼
2009-May	1100	Rev▲
2009-Jun	1400	Rev▲
2009-Jul	1600	Rev▲

Table 4.1: The relationship between two source measures and a target measure

4.2 The Sentinel Concept

Table 4.1 is an example, where two subsets have been extracted from a database. We have assigned short names to the measures as follows: *PeoInv* = the number of people involved in the decision process for customer projects, *UniRev* = the revenue of training courses, and *Rev* = revenue for the entire organization. The source measures, *PeoInv* and *UniRev*, have been extracted for January 2008 to April 2009. The target measure, *Rev*, has been extracted for April 2008 to July 2009; a similar period in length starting three months later. We refer to these three months as the *Warning Period*. We have shown the cases where measures change 10% or more, either up (▲) or down (▼), from one month to another.

As seen in the 16 rows in Table 4.1, the measures *PeoInv* and *UniRev* tend to change in a combined pattern such that when *PeoInv* goes up, *UniRev* goes down, and vice versa. This source measure pattern is observed 13 times, out of 15 possible. If we combine this pattern with the subsequent changes to *Rev* three months later, we see that *Rev* changes in the same direction as *UniRev* in 12, out of 13 possible times (denoted by $\#ChangesToSource = 13$). Another observation is that the relationship *Rev* and the combination of *PeoInv* and *UniRev* goes in both directions, which is a property we refer to as *bi-directionality*. Intuitively, one can say that if a relationship is bi-directional, then there is a greater chance that the relationship is causal, as opposed to a uni-directional relationship where a pattern is observed for measure changes in one direction only. Consider a case where revenue and staff costs increase

over a period of time. This yields the uni-directional relationship that an increase in revenue leads to an increase in staff costs the following month; in this case a decrease in revenue will not necessarily lead to a decrease in staff costs since these costs tend to be more fixed. Therefore, bi-directional relationships are more desirable. It is also noteworthy that *Rev* changes 6 times up (denoted by $A = 6$) and 6 times down (denoted by $B = 6$) in combination with *PeoInv* and *UniRev* since this "balance" again adds to the likeliness that the relationship is indeed causal. In summary we can say that a sentinel exists in Table 4.1 where changes in *PeoInv* and *UniRev* is able to warn three months ahead about changes to *Rev* with a *Confidence* of 92% (12 out of 13 times), defined as $Confidence = \frac{|A+B|}{\#ChangesToSource}$. $Balance = \frac{4\times|A|\times|B|}{(|A|+|B|)^2}$ is a measure for the degree to which a sentinel is balanced, and in this case the sentinel is perfectly balanced, meaning that $Balance = 1$. We note that the higher quality from bi-directionality achieved by assessing *Balance*, is impossible to achieve for sequential patterns since they can only represent one direction of changes in each pattern.

In addition to the combined relationship of the source measures, we can also observe "simple" sentinels (Chapter 2) with only one source and one target measure in Table 4.1. However, the *inverted* relationship between *PeoInv* and *Rev*, as well as the relationship between *UniRev* and *Rev*, each have one occurrence (the first two changes) where *Rev* changes in the opposite direction of what we would expect from all other changes. To assess the prediction ability for such sentinels we must first eliminate its *internal contradictions*. In this case, it is done by simply deducting the number of times *Rev* changes in the "unexpected" direction from the number of times *Rev* changes in the "expected" direction. This means that both source measures change 14 times, whereas the target measure after elimination changes only 11 times $(12 - 1)$. Therefore the simple sentinels have a poorer *Confidence* of 79% ($\frac{5+6}{14}$) and are slightly less balanced ($Balance = \frac{4\times|5|\times|6|}{(|5|+|6|)^2} = 0.99$) compared to the sentinel where the source measures were combined. On the other hand, simpler sentinels with fewer source measures have the advantage of being more general than very specific, potentially *overfitted*, sentinels with many source measures, and therefore the simplicity of a sentinel is also important.

4.3 The SentHiRPG algorithm

The *SentHiRPG* algorithm in Chapter 3 can find so-called generalized sentinel rules, like the sentinels found in the previous example. *SentHiRPG* applies a novel scoring principle, *Score* (Formula 4.1), which incorporates *Confidence*, *Balance*, *Warning Period*, and number of source measures of the sentinels.

$$
\begin{aligned}
Score = & \frac{|A+B|}{\max|A+B|} \times Confidence \times (\frac{1}{2} + \frac{Balance}{2}) \\
& \times (\frac{1}{2} + \frac{1 + MaxW - Warning\ Period}{MaxW \times 2}) \\
& \times (\frac{1}{2} + \frac{1 + MaxSource - |Source|}{MaxSource \times 2})
\end{aligned} \tag{4.1}
$$

Score expresses the quality of a sentinel as a single value and thus allows us to identify the best sentinel(s) in any larger set of sentinels. $\max|A+B|$ is found in the entire set of sentinels. *MaxW* and *MaxSource* are thresholds for the maximum *Warning Period* and the maximum number of source measures we are willing to accept. Using *Score*, the *SentHiRPG* applies a novel *Reduced Pattern Growth* (RPG) optimization that can quickly identify which measures that are candidates for the strongest relationships. RPG is facilitated by an intermediate optimized format called *The Table of Combinations* (TC). In addition, *SentHiRPG* applies a *hill-climbing* approach to find the best warning period for the sentinels.

The Table of Combinations is an intermediate hash table that is generated in one pass over the input data. Once generated, it can represent any measure change combination needed to mine all potential sentinels.

Reduced Pattern Growth delivers a good approximation of the top sentinels, and costs only 14% of the comparable cost for a full pattern growth (Chapter 3). The idea is to quickly identify the most *influential* source measures, where influence is defined as the number of times that a source measure change whenever the target measure also change. Having calculated the influence for all source measures, a Pareto Principle is applied to select the source measures in ranked order that account for 85% (configurable) of the sum of all influences. From this point, we *grow* sentinels from the measures identified. Starting with 1 source measure, we add the remaining influential source measures one at a time to create longer rules until the maximum number of source measures we desire is reached. In this process we only store a sentinel, and continue to add source measures, if the added source measures give a higher *Score*.

We note that the unique bi-directional nature of sentinels gives rise to the TC and RPG optimizations (Chapter 3), and thus cannot be offered by sequential pattern mining or other known optimizations for simpler "market basket"-type data such as [11].

Hill Climbing is used to identify the warning period, *w*, where the sentinel with the highest *Score* exists as an alternative to calculating all max(*Score*) for all *w*. Using a specialized 2-step/1-step hill climbing with two starting points, this approach only consume 53% of the time compared to testing all possible warning periods (Chapter 3).

The SentHiRPG algorithm can be described as three steps that incorporate these three optimizations:

Input: A target measure, a time-dimension, and a set of cubes.
Step 1: Build TC during one scan of the input data.
Step 2: **Hi**ll climb w to max(*Score*) of the sentinels constructed from TC with the source measures found in **RPG**.
Step 3: **Output** sentinels for w that meet the quality thresholds.

4.4 Sentinels in the TARGIT BI Suite

The architecture of the TARGIT BI Suite is shown in Figure 4.1. The implementation of sentinel mining in the TARGIT BI Suite consists of two parts: 1. The dialogue shown in Figures 4.3(a) and 4.3(b) which has been implemented in the TARGIT client, and 2. the sentinel engine in the TARGIT ANTserver (highlighted with red in Figure 4.1). The client dialogue allows the user to manipulate the input parameters before sending the input (target measure, time-dimension, and a set of cubes) to the sentinel engine. Upon completion of the mining process, the sentinel engine will transfer the sentinels found to the dialogue, that will then present these. From this stage the user can select one or more sentinels to become agents and submitted to the TARGIT ANTserver's scheduling service. The demonstration in this chapter will focus on the client dialogue, and from this perspective we will take the highly efficient *SentHiRPG* algorithm implemented in the TARGIT ANTserver for granted. A detailed description of the server side implementation and performance can be found in Chapter 5.

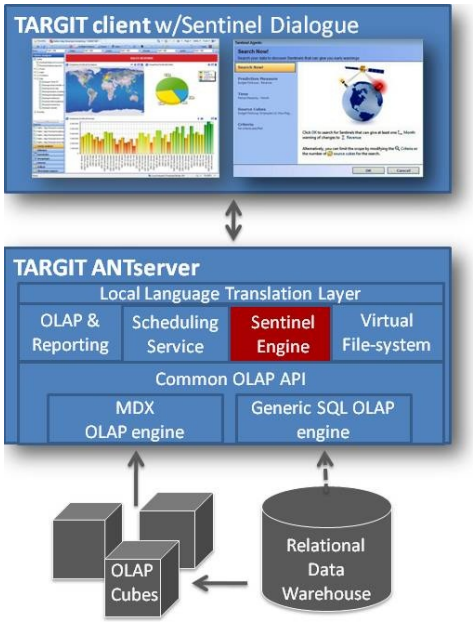


Figure 4.1: Architecture.

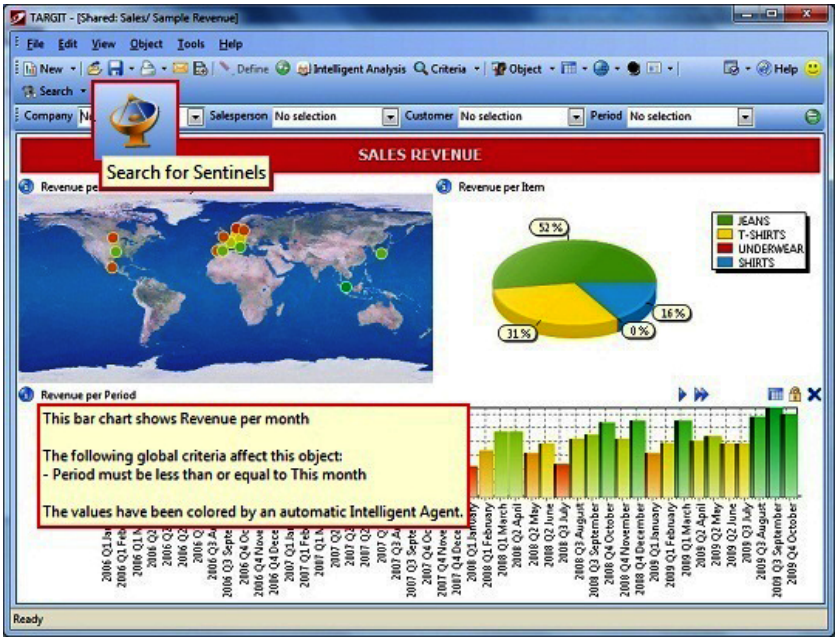


Figure 4.2: Context in an analysis in TARGIT BI Suite

A few words about clicks and context: As mentioned in Section 4.1, it is a big challenge to allow users to data mine without any prior technical knowledge. For this purpose we apply ”fewest clicks” as a quantitative approach to usability. The rationale is that minimizing the number of interactions (clicks) the user has during the OODA loop equals reducing the amount of training needed as well as the risk of making mistakes; and most importantly, we improve the speed of the cycle. To reduce the number of clicks, we keep track of the user's *context*. The context includes the measures and the dimension levels and criteria over which they are displayed, as well as the importance of a given measure compared to others, e.g., if a measure is analyzed on more dimension levels or more often than others, then it is most likely important. As shown in Figure 4.2, the context can be used to provide the user with an exact explanation about the data shown.

The context also allows the user to move very freely between the different BI disciplines, e.g., from a reporting context (characterized by a formalized layout) directly to an analysis context (characterized by displaying the data over more dimensions) with just one click. Similarly, the user can go directly from an analysis (or report) to a sentinel search simply by clicking the “search for sentinels” button (Figure 4.2) whenever something is interesting.

4.5 The Demonstration

The setup: The demonstration is run on the data warehouse in TARGIT A/S that has been operational for ten years. The data warehouse has been continuously adapted to the challenges of the organization. Today, the TARGIT data warehouse is based on a Microsoft (MS) SQL Server 2008 which is used for storage and staging of operational data. Upon transformation into cubes for different business areas, these data are made available to users through the TARGIT BI Suite that resides on top of the MS Analysis Services 2008. The data warehouse contains 16 GB of data, organized in 16 cubes with 250 measures and 109 dimensions. It is "mature" in the sense that there has not been any significant change to number of measures and dimensions over the past few years.

Searching for sentinels: The following demo flow will be shown live. However, the session will be interactive in order for users to see sentinel mining from other contexts as well.

Clicking the "search for sentinels" button (magnified) in Figure 4.2 will make the dialog in Figure 4.3(a) appear. We note that the system detects that revenue is most "interesting" over monthly periods since this was the context of Figure 4.2. From this point we can initiate the sentinel discovery process or use the dialog to change the Prediction Measure, the Source Cubes, Time, or the Criteria before clicking "OK". The search will typically run for a few minutes on the server before Figure 4.3(b) appears.

The sentinels in Figure 4.3(b) were found in the TARGIT data warehouse. The best sentinel is based on a combined relationship between the number of people involved in the decision process for customer projects and the revenue of training courses at the "TARGIT University". The direction in which the measure changes are related is shown by the red and green (dark and light in grey-scale) bi-directional arrows next to each of the measures. The top sentinel shows that if the number of people involved decrease and the university revenue increase, both by 10% or more, then the total revenue for TARGIT A/S is expected to increase by 10% or more within three months. As explained in Section 4.1, the sentinel is bi-directional and thus works in the opposite direction as well.

(a) **Start** by selecting the target measure and period for warnings

(b) **End** by listing the sentinels that can give an early warning

Figure 4.3: Searching for sentinels in the TARGIT BI Suite

By selecting this sentinel and clicking the "schedule" button, the user will now be notified with a given frequency, typically equal to the period over which we searched for sentinels (in this case monthly). If the combined incident of People Involved increase and University Revenue decrease occur, then the user will receive an email or notification directly on the desktop stating that:

```
Revenue is expected to decrease at least 10% in 3 month(s) because:
    People Involved has increased at least 10%
and
    University Revenue has decreased at least 10%.

The prediction has a confidence of 92%.
Click here to TARGIT the notification context,
or click here to review the Agent properties.
```

This means that the user will know three months ahead that something might happen to the overall revenue, and in addition, the user knows which measures to use as context in order to investigate what is causing the problem. At this point we say that the sentinel has contributed with synergy in the OODA loop since it alerted the attention very early to a problem that was most likely invisible to the user. In this particular case for TARGIT A/S, it was surprising that the number of people involved in the decision process could be used as an indicator, whereas it has been known for some time that selling more training will typically make a customer expand his solution. Intuitively, it does however make sense that the more people are involved in a decision process, the more time it will take, and therefore less revenue will be generated on the short-term; and vice versa. In other words, the users are now able to react faster if future revenue is threatened based on this new knowledge.

The TARGIT BI Suite also facilitates an even more radical approach when using sentinels. The "select all" option allows all sentinels to be scheduled as a "sentinel swarm". In this case the swarm will be monitoring everything that is going on in and around the organization, and report if something occurs, that seems to threaten a critical measure. Once a warning occurs the user will then decide what to do based on his orientation of the situation. Having a "sentinel swarm", rather than having only the sentinels that makes sense from a human perspective, appears to be an even more synergic approach to facilitating a fast OODA loop.

4.6 Conclusion

In this demo we presented the sentinel concept, a scoring principle, and its implementation in the TARGIT BI Suite. In addition, we presented the algorithm and data

warehouse setup that were prerequisites for our demo. We demonstrated a dialogue in our implementation where users, without any prior technical knowledge, are able to select a critical measure, a number of cubes, and a time dimension, and subsequently mine and schedule sentinels for *early warnings*. Sentinels was rated the most interesting and promising feature of TARGIT BI Suite version 2K9 in April 2009 by TARGIT partners representing a market footprint of 1,936 customers with more than 124,000 users. In addition, leading industry analyst, Gartner, introduced TARGIT in their Magic Quadrant for BI Platforms in 2010 and listed sentinels as one of the key strengths of TARGIT (Chapter 5).

Chapter 5

Implementing Sentinels in the TARGIT BI Suite

This chapter describes the implementation of so-called *sentinels* in the TARGIT BI Suite. Sentinels are a novel type of rules that can warn a user if one or more *measure changes* in a multi-dimensional data cube are expected to cause a change to another measure critical to the user. Sentinels notify users based on previous observations, e.g., that revenue might drop within two months if an increase in customer problems combined with a decrease in website traffic is observed. In this chapter we show how users, without any prior technical knowledge, can mine and use sentinels in the TARGIT BI Suite. We present in detail how sentinels are mined from data, and how sentinels are scored. We describe in detail how the sentinel mining algorithm is implemented in the TARGIT BI Suite, and show that our implementation is able to discover strong and useful sentinels that could not be found when using sequential pattern mining or correlation techniques. We demonstrate, through extensive experiments, that mining and usage of sentinels is feasible with good performance for the typical users on a real, operational data warehouse.

5.1 Introduction

Bringing data mining to the masses have been a quest by major business intelligence (BI) vendors since the late 1990s [50]. However, a decade later this "Next Big Thing" is yet to happen according to OLAP industry analyst Nigel Pendse, author of the OLAP Report [50]. According to the most popular interpretation of Moore's Law, we can say that within a three year period the computing performance available to

an organization will have increased by 400%, during the same period the amounts of structured business data in an organization does not grow more than about 95% (20-30% per year) according to industry analysts IDC[1]. This means that a significant amount of computing capacity is available to analyzing the measures and dimensions in any data warehouse. Therefore, there is an obvious potential to use these computing resources to allow users in an OLAP environment to use data mining for exploring data relationships that are practically impossible to find manually.

We believe that integration of BI disciplines and usability is the key to unlock the big potential of end user data mining that has not yet reached the business users. With this in mind, we have implemented so-called *sentinels* in the commercial TARGIT BI Suite, available to 274,000 users in 3,800 organizations world-wide.

A sentinel is a novel type of causal rule-based relationship; the concept and formal definitions have been developed in a collaborative research project between TARGIT A/S and Aalborg University (Chapters 2 and 3). Sentinels are discovered through a data mining process, where changes in one or multiple *source measures* are followed by changes to a *target measure* (typically a KPI), within a given time period, referred to as the *warning period*. An example of a sentinel for a company could be: "IF Number of Customer Problems go up and Website Traffic Volume goes down THEN Revenue goes down within two months AND IF Number of Customer Problems go down and Website Traffic Volume goes up THEN Revenue goes up within two months". Such a rule will allow a BI system to notify a user to take corrective action once there is an occurrence of, e.g., "Number of Customer Problems go up and Website Traffic Volume goes down". The decision maker is now able to respond faster based on a threat to his KPI (Revenue), a threat that might be invisible to him without sentinels. In the TARGIT BI Suite it is now possible for any user to find such sentinels in any given *context*. Subsequently, it is possible to use the sentinels to provide early warnings if critical business goals are threatened. We note that the user only needs to know the KPI, and thus no data mining knowledge is required.

Sentinels are based on the Computer Aided Leadership & Management (CALM) theory [36]. The idea in CALM is to take the Observation-Orientation-Decision-Action (OODA) loop (originally pioneered by "Top Gun" fighter pilot John Boyd in the 1950s), and integrate BI technologies to drastically increase the speed with which a user "travels" through the OODA loop. Sentinels improve the speed of the OODA loop's observation and orientation phases by giving the decision maker an early warning (faster observation) that threatens a KPI. At the same time, the sentinel highlights the threat (faster orientation) by listing the measure changes that appears to be "causing" it. In other words, sentinels contribute with both synergy and efficiency for a user cycling an OODA loop.

[1] ComputerWorld, September 22nd, 2009, BI expert, Brian Troelsen, IDC.

Sentinels are mined on the measures and dimensions of multiple cubes in an OLAP database, as opposed to the "flat file" formats used by most traditional data mining methods. Sentinels find rules that would be impossible to detect using traditional techniques such as sequential pattern mining and correlation techniques (Chapter 2). As explained in detail in Section 5.6, the *bi-directional* sentinel rules are *stronger* rules than those mined by sequential pattern and gradual rule mining. In addition, sentinels are more *specific* than the relationships that can be found using regression techniques.

In the following section, we present the in-depth motivation for sentinels along with a real world case of sentinel application. Section 5.3 presents the implementation of sentinels in the TARGIT BI Suite, a commercial software package from the company TARGIT A/S. In Section 5.4 we present our real-world experience with sentinels in the market since their commercial availability in April 2009. In Section 5.5 we present experiments of the sentinel discovery process on large amounts of data. Section 5.6 presents the work related related to sentinels, and Section 5.7 presents our conclusions and proposals for future work.

5.2 Motivation & Case

The TARGIT case: The data warehouse in TARGIT A/S has been operational for ten years, and it has been continuously adapted to the challenges of the organization. Today, the TARGIT data warehouse is based on a Microsoft (MS) SQL Server 2008 which is used for storage and staging of operational data. Upon transformation into cubes for different business areas, these data are made available to users through the TARGIT BI Suite that resides on top of a number of MS Analysis Services 2008 cubes. The data warehouse contains 16 GB of data, organized in 16 cubes with 250 measures and 109 dimensions. It is "mature" in the sense that there has not been any significant change to number of measures and dimensions over the past few years.

The data warehouse covers data from all business areas within TARGIT A/S, i.e., sales, development, support, and administration. Since all the data are integrated, this means that the data warehouse covers a complete life-cycle for all TARGIT customers, e.g., we know how much software they own, how much training they got, how many problems they had, and which suggestions they have for the future versions of the TARGIT software. In addition, the TARGIT data warehouse covers information about the decision process from the showing of interest in TARGIT's software to becoming a customer. Based on this information, TARGIT A/S is navigating in global competition through a network of more than 280 partners (resellers/system integrators) world-wide. In this respect, it is also possible for the partners to access part of the data warehouse to optimize their ability to sell and support TARGIT's software.

(a) **Start** by selecting the target measure and period for warnings

(b) **End** by listing the sentinels that can give an early warning

Figure 5.1: Searching for sentinels in the TARGIT BI Suite

A user can access the data warehouse through a Windows based client that connects directly to the TARGIT ANTserver, or through a browser that renders the zero footprint client of the TARGIT NET server that then connects to the TARGIT ANTserver to access the data (See Section 5.3).

The TARGIT case is used in the current section to demonstrate the implementation of sentinels in the TARGIT software. In addition, the "real data" performance studies in Section 5.3 are based on this case. Finally, whenever we refer to specific numbers of TARGIT customers, users, and partners in this chapter, these figures have been extracted from the TARGIT data warehouse on December 17th 2010.

A few words about clicks and context: It is a big challenge to allow users to data mine without any prior technical knowledge. For this purpose we apply "fewest clicks" as a quantitative approach to usability. The rationale is that minimizing the number of interactions (clicks) the user has during the OODA loop equals reducing the amount of training needed as well as the risk of making mistakes; and most importantly, we improve the speed of the cycle. To reduce the number of clicks, we keep track of the user's *context*. The context includes the measures and the dimension levels and criteria over which they are displayed, as well as the importance of a given measure compared to others, e.g., if a measure is analyzed on more dimension levels or more often than others, then it is most likely important.

The context also allows the user to move very freely between the different BI disciplines, e.g., from a reporting context (characterized by a formalized layout) directly to an analysis context (characterized by displaying the data over more dimensions) with just one click. In other words, the user can move from the observation to the orientation phase with just one click. Using the context, the user can search for sentinels very intuitively simply by clicking "search for sentinels" whenever he finds something interesting.

Searching for sentinels – user perspective: In Figure 5.1(a) we see the initial dialogue for the sentinel search. The dialog has been launched from an analytical context where the measure revenue was shown over three different dimensions: *Time (Month)*, *Geography*, and *Product*. We note that the system detects that revenue is most "interesting" over monthly periods since this was the context the user was in before. From this point it is possible to initiate the sentinel discovery process (Chapter 3) or use the dialog to change the Prediction Measure, the Source Cubes, Time, or the Criteria. By proceeding, the search will typically run for a few minutes on the server before Figure 5.1(b) appears. The run-time is primarily influenced by the number of measures in the data warehouse as seen in the performance study in Section 5.5.

The sentinels in Figure 5.1(b) were found in the TARGIT data warehouse. The best sentinel is based on a combined relationship between the number of people involved in the decision process for customer projects and the revenue of training courses at the "TARGIT University". The direction in which the measure changes are related is shown by the red and green (dark and light in grey-scale) bi-directional arrows next to each of the measures. The top sentinel shows that if the number of people involved decrease and the TARGIT university revenue increase, both by 10% or more, then the total revenue for TARGIT A/S is expected to increase by 10% or more within three months. The sentinel is bi-directional and thus works in the opposite direction as well. By scheduling this sentinel for notification, the user will now be notified with a given frequency, typically equal to the period over which we searched for sentinels (in this case monthly). If the combined incident of "People Involved" increase and "University Revenue" decrease occurs, then the user will receive an email or notification directly on the desktop stating that:

```
Revenue is expected to decrease at least 10% in 3 month(s) because:
   People Involved has increased at least 10%
and
   University Revenue has decreased at least 10%.

The prediction has a confidence of 92%.
Click here to TARGIT the notification context,
or click here to review the Agent properties.
```

This means that the user will know three months ahead that something might happen to the overall revenue, and in addition, the user knows which measures to use as context in order to investigate what is causing the problem. At this point we say that the sentinel has contributed with *synergy* in the OODA loop since it alerted the attention very early to a problem that was most likely invisible to the user. In this particular case for TARGIT A/S, it was surprising that the number of people involved in the decision process could be used as an indicator, whereas it has been known for some time that selling more training will typically make a customer expand his solution. Intuitively, it does however make sense that the more people are involved in a decision process, the more time it will take, and therefore less revenue will be generated on the short-term; and vice versa. In other words, the users are now able to react faster if future revenue is threatened based on this new knowledge.

The TARGIT BI Suite facilitates an even more radical "select all" option, that schedules all sentinels in a "sentinel swarm". In this case the swarm will be monitoring everything that is going on in and around the organization, and report if something occurs, that seems to threaten a critical measure. Once a warning occurs the

user will then decide what to do based on his orientation of the situation. Having a "sentinel swarm", rather than having only the sentinel rules that makes sense from a human perspective, appears to be an even more synergic approach to facilitating a fast OODA loop.

Once one or more sentinels have been scheduled for notification, and the users start reacting upon these, the confidence of the sentinels will most likely change. The reason is, that the user interferes with the "history" of causality, e.g., if a user successfully addresses the challenge posed by "People Involved" increasing and "University Revenue" decreasing during the next three months, and thereby avoid that "Revenue" decrease, then the confidence of the sentinel itself will decrease. Confidence and other qualitative measures for sentinels are described below. From a user perspective it should be noted that the confidence of a sentinel is fluid, and depending on users seeking to avoid or fulfill the "prediction" of the sentinel, confidence will change based on the users actions. If the confidence of a sentinel changes below certain thresholds (see Section 5.3) set in the TARGIT BI Suite, the user will be notified, but this time the sentinel will suggest that it is removed from the active notifications list. This way the users can manage a fluid set of notifications where newly mined sentinels are scheduled, and irrelevant sentinels are retired.

It should be noted, that a sentinel mining process does not need to be conducted while the user is online. However, as we will see in the experiments in Section 5.5, the sentinel mining process can many times be conducted with a few minutes as the maximum response time. Therefore sentinel mining is also very feasible in an environment where users want to stay online while conducting their mining processes.

Searching for sentinels – data perspective: To exemplify the sentinel mining process that takes place, a data example is presented in Table 5.1(a) and 5.1(b), where two subsets have been extracted from an example database. Please note that for reasons of confidentiality, the example provided is *not* real data from the TARGIT data warehouse. The source measures have been extracted for January 2008 to April 2009. The target measure has been extracted for April 2008 to July 2009; in other words, for a similar period in length starting three months later. For both source and target measures we have calculated the cases where a measure changes 10% or more, either up (▲) or down (▼), from one month to another. For easy reference, we have assigned short names to the measures as follows: *PeoInv* = "People Involved", *UniRev* = "University Revenue", and *Rev* = "Revenue".

As seen in the 16 rows in Table 5.1, the measures *PeoInv* and *UniRev* tend to change in a combined pattern such that when *PeoInv* goes up, *UniRev* goes down, and vice versa. This pattern in the source measures is observed 13 times, out of 15 possible. If we combine this pattern with the subsequent changes to *Rev* three months later, we see that *Rev* changes in the same direction as *UniRev* in 12, out of

(a) Source

Month	*PeoInv*	*UniRev*	Change
2008-Jan	1	115	
2008-Feb	2	115	$PeoInv$▲
2008-Mar	2	100	$UniRev$▼
2008-Apr	3	90	$PeoInv$▲,$UniRev$▼
2008-May	2	363	$PeoInv$▼,$UniRev$▲
2008-Jun	3	310	$PeoInv$▲,$UniRev$▼
2008-Jul	2	440	$PeoInv$▼,$UniRev$▲
2008-Aug	4	297	$PeoInv$▲,$UniRev$▼
2008-Sep	5	260	$PeoInv$▲,$UniRev$▼
2008-Oct	6	230	$PeoInv$▲,$UniRev$▼
2008-Nov	4	294	$PeoInv$▼,$UniRev$▲
2008-Dec	5	264	$PeoInv$▲,$UniRev$▼
2009-Jan	6	230	$PeoInv$▲,$UniRev$▼
2009-Feb	4	270	$PeoInv$▼,$UniRev$▲
2009-Mar	3	353	$PeoInv$▼,$UniRev$▲
2009-Apr	2	400	$PeoInv$▼,$UniRev$▲

(b) Target

Month	*Rev*	Change
2008-Apr	900	
2008-May	1001	Rev▲
2008-Jun	1200	Rev▲
2008-Jul	750	Rev▼
2008-Aug	1001	Rev▲
2008-Sep	1100	
2008-Oct	1250	Rev▲
2008-Nov	970	Rev▼
2008-Dec	850	Rev▼
2009-Jan	720	Rev▼
2009-Feb	1250	Rev▲
2009-Mar	930	Rev▼
2009-Apr	800	Rev▼
2009-May	1100	Rev▲
2009-Jun	1400	Rev▲
2009-Jul	1600	Rev▲

Table 5.1: The relationship between two source measures and a target measure

13 possible times. Another observation is that the relationship *Rev* and the combination of *PeoInv* and *UniRev* goes in both directions, which is a property we refer to as *bi-directionality*. Intuitively, one can say that if a relationship is bi-directional, then there is a greater chance that the relationship is causal, as opposed to a uni-directional relationship where a pattern is observed for measure changes in one direction only. Consider a case where revenue and staff costs increase over a period of time. This yields the uni-directional relationship that an increase in revenue leads to an increase in staff costs the following month; in this case a decrease in revenue will not necessarily lead to a decrease in staff costs since these costs tend to be more fixed. Therefore, bi-directional sentinels are more desirable. In the example, it is noteworthy that *Rev* changes 6 times up and 6 times down in combination with *PeoInv* and *UniRev* since this "balance" again adds to the likeliness that the relationship is indeed causal. We say that a perfectly balanced sentinel exists in Table 5.1, where changes in *PeoInv* and *UniRev* is able to predict changes three months later in *Rev* with a historical accuracy of 92% (12 out of 13 times).

In addition to the combined relationship of the source measures, we can also observe "simple" sentinels with only one source and one target measure in Table 5.1. However, the *inverted* relationship between *PeoInv* and *Rev*, as well as the relationship between *UniRev* and *Rev*, each have one occurrence (the first two changes) where *Rev* changes in the opposite direction of what we would expect from all other changes. To assess the ability to predict for such sentinels we must first eliminate its internal contradictions. In this case, it is done by simply deducting the number

of times *Rev* changes in the "unexpected" direction from the number of times *Rev* changes in the "expected" direction. This means that both source measures change 14 times, whereas the target measure after elimination changes only 11 times $(12-1)$. Therefore the simple sentinels have a poorer historical accuracy of 79% (11 out of 14 times) compared to the sentinel where the source measures were combined. On the other hand, simpler sentinels with fewer source measures have the advantage of being more general than very specific, overfitted sentinels with many source measures, and therefore sentinel simplicity is also important.

Distinguishing good sentinels from others: The formal definitions of the sentinels in TARGIT BI Suite can be found in Chapter 3. However, in order to give an intuitive idea of the properties that distinguishes a good sentinel rule from other rules, we provide the following short definition for the quality of a sentinel. The sentinel, denoted by $Source \rightsquigarrow Target$, is based on a set of source measures, *Source*, and a target measure, *Target*, and we have a warning period, w, which passes between a change in *Source* to a change in *Target*.

In Formulae 5.1 to 5.3, *A* is the number of times that *Source* changes in one direction and *Target* subsequently changes in the "expected" direction, minus the number of times where *Target* did not change in the expected direction. *B* is calculated in the same way, but for the changes to *Source* in the opposite direction of *A*. For simplicity, we assume that $|A| \times |B| > 0$ since we are only interested in *bi-directional sentinels* in this implementation (see explanation for Formula 5.3 below). A general definition of Formulae 5.2 and 5.3 can be found in Chapter 3. *Balance* is used to determine the degree to which a sentinel is uni-directional (*Balance=0*) or completely bi-directional (*Balance=1*), meaning that the sentinel indicates a decrease to the target measure exactly as many times as it indicates an increase. *Conf* tells us how often, when a change in the source measure(s) occurs, the expected change in the target measure subsequently occurs within w time.

$$Balance_{Source \rightsquigarrow Target} = \frac{4 \times |A| \times |B|}{(|A| + |B|)^2} \tag{5.1}$$

$$Conf_{Source \rightsquigarrow Target} = \frac{|A + B|}{\text{Number of changes to } Source} \tag{5.2}$$

$$
\begin{aligned}
Score_{Source \rightsquigarrow Target} = & \\
& \frac{|A+B|}{\text{Highest } |A+B| \text{ found in sentinels compared}} \\
& \times Conf_{Source \rightsquigarrow Target} \\
& \times (\frac{1}{2} + \frac{Balance_{Source \rightsquigarrow Target}}{2}) \\
& \times (1 - wp + \frac{(1 + Maxw - w) \times wp}{Maxw}) \\
& \times (\frac{1}{2} + \frac{1 + MaxSource - |Source|}{MaxSource \times 2})
\end{aligned}
\tag{5.3}
$$

Formulae 5.1 and 5.2 each represent one quality of a sentinel, but in order to have one single value to determine which sentinel is the best, we introduce *Score* as shown in Formula 5.3. *Score* takes into consideration the actual number of times there is support for a sentinel compared to other sentinels in a cube (normalized into $]0, ..., 1]$), adjusted for contradictions, $|A + B|$, as well as the confidence, *Conf*, and *Balance* of the sentinel. It is desirable that a sentinel has a high number of occurrences as well as a high confidence. However, the balance of the sentinel is also important since it allows us to identify bi-directional sentinels, since bi-directional sentinels in general have a higher probability of being causal as opposed to coincidental. In addition, we introduce the threshold, *Maxw*, which is the maximum length of the warning period, *w*, we are willing to accept. The constant, *wp*, represents the warning penalty, i.e., the degree to which we want to penalize sentinels with a higher *w* (0=no penalty, 1=full penalty). The idea of penalizing higher values of *w* is relevant if a pattern is cyclic, e.g., if the indication of a sentinel occurs every 12 months, and the relationship between the indications on the source measure(s) and the target measure is less than 12 months, then the a given sentinel with a warning period *w* is more desirable than the same sentinel with a warning period *w*+12. We also take into consideration that it is desirable to have shorter, general rules, meaning a low cardinality of *Source*. This prevents our implementation from "overfitting" rules [45] and thus generating very specific and therefore irrelevant rules. In other words, Formula 5.3 will yield a better *Score* when as many as possible of the following conditions apply simultaneously:

- $|A + B|$ is as high as possible, meaning that many changes in the source measures are followed by the "expected" changes in the target measure
- A and B are equal or close to equal, meaning that a bi-directional sentinel is preferred
- w is as short as possible, meaning that sentinels with short warning periods are preferred to sentinels with longer warning periods

- the are as few source measures as possible, meaning that the sentinel is as general as possible

To exemplify, the calculation of *Score* let us review the best sentinel shown in the top on Figure 5.1(b). The sentinel indicates that Revenue will change at least 10% three months after People Involved and University Revenue have changed at least 10% in accordance with the direction of the arrows. In this case it was found in the data (Table 5.1) that a decrease in People Involved combined with an increase in University Revenue occurred 6 times where Revenue subsequently increased. Moreover, the exact opposite scenario of an increase in People Involved combined with a decrease in University Revenue also occurred 6 times, and here Revenue subsequently decreased. This gives us a completely balanced rule ($Balance = \frac{4\times6\times6}{(6+6)^2} = \frac{144}{144} = 100\%$) which is a sign of a highly causal relationship between the measures, as opposed to a more coincidental relationship, e.g., if the rule had a tendency to work only in one direction. The changes to the source measures occur one time without the expected impact on Revenue, i.e., the change pattern on the source measures occurs in one or the other direction 13 times in total. This means that the sentinel has a confidence of 92% ($\frac{6+6}{13} = \frac{12}{13}$). Using a maximum of 3 source measures allowed ($MaxSource = 3$), a maximum allowed warning period of 12 months ($Maxw = 12$), and a warning penalty of 0.5 ($Wp = 0.5$) we have a *Score* of 0.60 ($\frac{12}{14} \times 0.92 \times (\frac{1}{2} + \frac{1}{2}) \times (1 - 0.5 + \frac{(1+12-3)\times0.5}{12}) \times (\frac{1}{2} + \frac{1+3-2}{3\times2})$). In the input for this *Score* it should be noted that the highest $|A + B|$ for any sentinel found at these thresholds was 14.

Using *Score*, we can select the optimal threshold for changes (referred to as α in Chapter 3) as well as the optimal warning period, w. In principle this is done by testing all combinations of α and w, and subsequently selecting the combination where the sentinel with the highest *Score* exist. However, in reality we can disregards some combinations without testing as explained in Section 5.3. In addition to fitting these values optimally, *Score* is used to rank the sentinels found in the output for the user. The sentinels shown in Figure 5.1(b) have been ordered by their respective *Score*. Furthermore, with reference to Chapter 3, *Score* also plays a major role in the optimization techniques applied to mine the sentinels efficiently. Finally, *Score* assists the casual users by providing a simple and uniform way to assess the sentinel quality. As mentioned in Section 5.2, it is the primary goal of TARGIT to empower the casual users with powerful analytics. However, for more advanced users, it is possible to toggle the default constants in *Score* and the thresholds for *Balance* and *Conf*.

5.3 Implementation

Architecture of the TARGIT BI Suite: The TARGIT BI Suite is a client/server based application which allows users to create reports, analysis, and dashboards (scorecards). In addition, the application allows users to search for sentinels and to schedule these for early warnings as explained in Section 5.2. The user connects a Windows based client program to a centralized TARGIT ANTserver through TCP/IP as shown in Figure 5.2. The TARGIT ANTserver has a number of services that can be offered to the client; a core functionality of these is to parse and forward a data query to either a cube or a relational database, and subsequently return the answer to the client. An alternative web client is also available. In this case the browser will render a "zero footprint" instance from a TARGIT NET server which holds the execution of the client. The TARGIT NET server will then connect to the TARGIT ANTserver similarly to the stand alone client shown in Figure 5.2. The TARGIT ANTserver is the backbone of the application. As shown in Figure 5.2, there are two general services (G1,G2) and four specific services (S1...S4) offered by the TARGIT ANTserver:

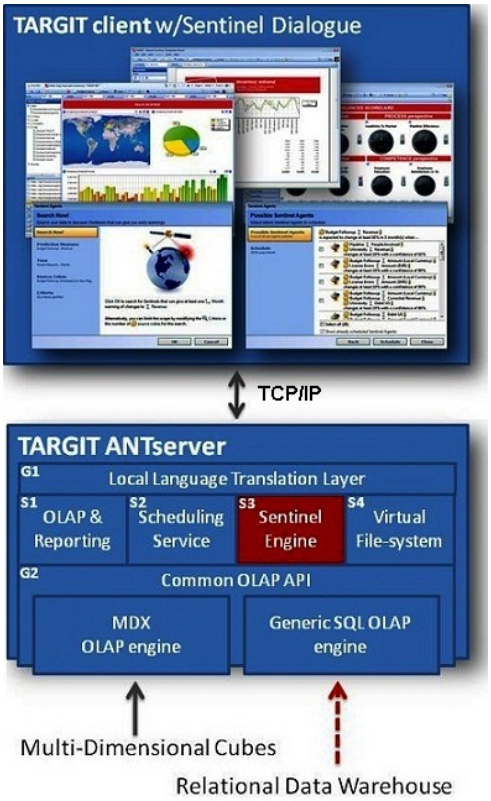


Figure 5.2: Architecture.

Local Language Translation Layer (G1) translates dimension and measure names as well as names for files shared among users. This layer also provides the language for the client application. Since the client program receives all language settings from the TARGIT ANTserver based on the user preference means that different users can access the same client program on a physical PC in different languages.

Common OLAP API (G2): This service facilitates that the functionality of the TARGIT ANTserver exceeds traditional MDX and SQL, e.g., the TARGIT ANTserver can run queries across multiple heterogeneous data sources and combine the answers. In addition, it has ex-

tended capabilities for handling time-dimensions dynamically, e.g., it allows criteria such as "time must be equal to this month" and other generic time variants, e.g., This year, previous year, previous month. Most importantly, the Common OLAP API allows all other services to treat the entire set of data sources as one homogenous cube with one unified interface.

OLAP & Reporting (S1): This service parses the incoming analytical and reporting queries in XML format from the client and submits them to the Common OLAP API. The results may require post-processing for additional formatting as stated in the XML, and upon completion the result is submitted in binary format to the client. The binary format for the response has been selected to optimize performance.

Scheduling Service (S2): This service facilitates that some jobs can be stored and executed at certain times. The jobs include reports and analysis as well as more general "slideshows" and "podcasts" of the information from the databases. The service also allows agents that monitor specific information to be scheduled, and with a predefined frequency the ANTserver will verify if the premise for notification of the user is met. If so, the user will be notified via email or directly via a special desktop notification client. One type of such agents are the sentinel agents that notify the users if the premise of source measure changes are met.

Sentinel Engine (S3): This new service facilitates the search for sentinels once the client dialogue on Figure 5.1(a) is completed. From this point, the SentHiRPG algorithm (Chapter 3), described below, performs the search for sentinels. Once the search is completed, the answer is returned to the client as seen in Figure 5.1(b), where it is possible to submit the sentinels found to the scheduling service (S2).

Virtual File-System (S4): This service allows users to store files (reports, analysis, and dashboards) in a repository where they can be shared with other users. The service also facilitates that Windows client files are available on the zero footprint web client through the TARGIT NET server.

The SentHiRPG Algorithm: The *SentHiRPG* algorithm (Chapter 3) can find so-called generalized sentinel rules in which multiple source measures are combined into strong, and most likely, causal relationships with a given target measure. *SentHiRPG* applies a novel *Reduced Pattern Growth* (RPG) optimization that quickly identify which measures that are candidates for the strongest relationships. RPG is facilitated by an intermediate optimized format called *The Table of Combinations* (TC). In addition, *SentHiRPG* applies a *hill-climbing* approach to fit the best sentinel warning period.

The Table of Combinations is an intermediate hash table that is generated in one pass over the input data, and used for optimization. Once generated, the TC represents exactly the measure change combinations needed to mine all potential sentinels. A reliable sentinel will require at least two, but typically multiple rows in the

TC, and since we do not know which source measure changes that occur at the same time, there is no generic sorting method that can optimize the scans further than the reduction of data in the TC.

Reduced Pattern Growth delivers a good approximation of the top sentinel rules, and it is much more efficient than a full pattern growth of all combinations of source measures. The idea is to first very quickly identify the source measures that change whenever the target measure change; at this stage we do not attempt to find out what type of relationship the source measures have to the target measure or to each other. For example, if we look at Table 5.1, we see that both "People Involved" and "University Revenue" have 13 occurrences where they change while "Revenue" also change. In this case we say that both these source measures have an *influence* of 13, and we note that there is no guarantee that a high influence will lead to a high *Confidence*, *Balance*, or *Score*. However, a low influence would most likely mean a low *Confidence* and subsequently a low *Score*, since the given source measure would then change very little in combination with the target measure. Once the influence for all source measures have been calculated, they are ranked, and a Pareto Principle is applied to select the source measures that account for a certain percentage (*RPGpareto*) of the sum of all influences. For example, let us assume that we have two additional source measures, "Staff Salaries", representing the salaries paid to employees, and "Product Quality", representing the number of errors found in the products sold. The source measure "Staff Salaries" has an influence of 2, and the source measure "Product Quality" has an influence of 3. The sum of all influences for the four source measures is then $13 + 13 + 3 + 2 = 31$ (note that the influences are ordered descendingly). If the value of *RPGpareto* is set to 85%, we need to identify the most influential source measures that account at least 85% of all influence. Starting with the highest influence observed, 13, we find that the two source measures, "People Involved" and "University Revenue" account for $\frac{13+13}{31} = 84\% < RPGpareto$. In this case we need to add the influence of the second highest influential source measure to account for at least 85% of all influence, $\frac{13+13+3}{31} = 94\% \geqq RPGpareto$. In this example, the process would only eliminate one out of four source measures. However, from experiments on real-world data, we know that the influence of source measures can be described as a *power law*, meaning that a few source measures have a high influence and are thus very likely to be part of many good sentinels, whereas the majority of source measures are not likely to be part of any good rule at all.

Having identified the most influential source measures, we *grow* sentinels from these measures. Starting with 1 source measure, we add the remaining influential source measures one at a time to create longer rules until the number of source measures equals *MaxSource*. During this process we only store a sentinel, and continue to add source measures, if the added source measures translates into a higher *Score*. For example, using the data in Table 5.1 we would first create two sentinels with

only one source measure, respectively "People Involved" and "University Revenue". Using the calculation demonstrated in Section 5.2, both these sentinels have a *Score* of 0.56 ($\frac{11}{14} \times 0.79 \times (\frac{1}{2} + \frac{0.99}{2}) \times (1 - 0.5 + \frac{(1+12-3)\times 0.5}{12}) \times (\frac{1}{2} + \frac{1+3-1}{3\times 2})$). If we combine the two source measures into a longer rule, we have a *Score* of 0.60, and thus the new longer rule will "survive". Subsequently, we would combine this sentinel with the remaining influential source measure, "Product Quality", to see if *Score* improved even more, and so on.

On a realistic dataset, the quality of the RPG approximation has been experimentally tested to produce 100% of the top 10 sentinels, and 88% of the top 100 sentinels that would otherwise have to be found by a full pattern growth, i.e., testing all combinations of source measure relationships with the target measure. The performance cost of RPG is only 14% of the comparable cost for a full pattern growth (Chapter 3).

Hill Climbing is used to auto-fit of the warning period. This is done by identifying the warning period, *w*, where the sentinel with the highest *Score* exists. Alternatively, all *Scores* for all sentinels for a each value of *w* would have to be inspected to find the sentinel with maximum *Score*. Using a specialized 2-step/1-step hill climbing with two starting points, this approach only consumed 53% of the time compared to testing all possible warning periods (Chapter 3).

The SentHiRPG algorithm can now be described as three steps:

Step 1: Build TC during one scan of the input data.
Step 2: **Hi**ll climb *w* to max(*Score*) of sentinels constructed
from TC with source measures found in **RPG**.
Step 3: Output sentinels for *w* that meet the quality thresholds.

Implementation of Sentinel Mining: The implementation of sentinel mining in the TARGIT BI Suite consists of two parts: 1. The dialogue shown in Figures 5.1(a) and 5.1(b) which has been implemented in the TARGIT client, and 2. the sentinel engine in the TARGIT ANTserver (S3 in Figure 5.2). The client dialogue allows the user to manipulate the input parameters before sending the "sentinel mining query" to the sentinel engine. Upon completion of the mining process, the sentinel engine will transfer the sentinels found to the dialogue presenting them. From this stage the user can select one or more sentinels to become agents and submitted to the TARGIT ANTserver's scheduling service (S2 in Figure 5.2). Since the dialogue on the client side is a simple matter of manipulating parameters and presenting output, we will focus on the implementation of the sentinel engine in the ANTserver.

The process implemented in the sentinel engine is shown in pseudo code as the *Sentinel Mining* function below. The declaration of the interface for the *SentHiRPG* function is strictly aligned with the definition in Chapter 3. With this prerequisite, sentinel mining can be described in three steps as follows:

Function: Sentinel Mining
Input: A target measure, *TM*, a shared time-dimension level, *T*, a set of cubes, *CubeSet*, a set of criteria on dimension members, *SliceCriteria*, and a max number of sentinels to be returned, *X*.
Output: Sentinels with a given warning period, w, a threshold for indications, α, and their respective *Conf* and *Score*.
Method: The sentinels are mined as follows:

Sub-Function: *SentHiRPG*
Input: A list of facts from a cube, *C*, ordered by $(d_2, d_3, ..., d_n, t)$, an offset, o, a maximum warning period length, *Maxw*, a maximum number of source measures per rule, *MaxSource*, a warning penalty, *wp*, a threshold for RPG, *RPGpareto*, a threshold for indications, α, a minimum *SentSupp* threshold, σ, a minimum *Conf* threshold, γ, and a minimum *Balance* threshold, β.
Output: Sentinel rules with a given warning period, *Optimalw*, and their respective *SentSupp*, *Conf*, *Balance*, and *Score*.
Method: The algorithm is described in previous section above.

Step 1: Using the Common OLAP API, all "pure" measures from the cubes $\in CubeSet$ are identified as source measures, *SM*, with the exception of *TM*. The facts to be mined, C, are extracted with one dimension (level), *T*, and the measures *SM* and *TM* for all $T <$ *current period of* T. If $SliceCriteria \neq \emptyset$ then each element is applied as a selection criteria to the cubes where it is possible. *Maxw* is set to the maximum cardinality of the children sets at the same level as T. The remaining parameters are set as follows: $o = 1$, $MaxSource = 3$, $wp = 0.5$, $RPGpareto = 85\%$, $\sigma = 5$, $\gamma = 80\%$, $\beta = 80\%$, and $X = 200$. These settings are based on practical experiences with real world data.
Step 2: Repeat *SentHiRPG* for each $\alpha \in \{10\%, 15\%, 20\%\}$, all other parameters remain constant as set in step 1. While testing different values of α, the set of sentinels for a given value of α, that contains the sentinel with the highest *Score* is stored in memory. This set is not flushed from memory until a sentinel with a higher *Score* exists in another set for a new value of α.
Step 3: Output the top *X* sentinels from memory to client through the Local Language Translation Layer. If the number of sentinels $< X$ then return all.

Step 1: We use the Common OLAP API to find all measures from all the cubes in *CubeSet* selected by the user. In this context we note that a relational database will also appear to be a cube in this context since it will have been mapped into dimensions and measures when it was plugged into the Common OLAP API by an administrator of the data warehouse. The time-dimension, *T*, is a shared dimension that allows the Common OLAP API to relate the measures collected to each other.

The dialogue in the client will only present cubes for selection that include this shared time-dimension. During the identification of all measures we seek to disregard measures that are only replicas of other *base* measures. If for instance a measure is calculated based on only one other measure, we will disregard the calculated measure and stick with the "base" measure which it was based on. Logically we will also disregard measures calculated on measures that are disregarded. By disregarding measures that are not base, we seek to eliminate measures that do not contribute with something new, and thereby we reduce the number of measures that needs to be mined.

We apply one general criteria on the time-dimension, *T*, that seeks to ensure that we do not mine on an incomplete period. This criteria means that we will only mine all periods prior to the one we are in, e.g., if we are currently on November 15th, we will only be mining all data up until and including October which is the last complete month. The same principle applies to whatever period we can think of, e.g., hour, week, or year. In addition, we apply the additional slicing from *SliceCriteria* if such has been specified. However, in this context we only apply these criteria where it is possible since the slicing members do not need to be from shared dimensions. This means that only cubes where it is possible to say "data must be from United States only" gets this criteria applied. One example of this could be that revenue can be selected for United States only, whereas the currency rate of Euros cannot. Nevertheless, there might be an interesting relationship between the revenue in United States and the Euro rate anyways. Therefore we are more loose in applying these criteria than with the strict shared time-dimension which is needed to relate and stage data for Step 2.

Setting *Maxw* to the maximum cardinality found in the children sets at the same level as T means that *Maxw* will never exceed the number of periods that is possible for a parent in the hierarchy of T. If T is on month level and we are currently in November, and a complete previous year of data exists, then $Maxw = 12$. If no previous year exists then $Maxw = 11$. In general, this means, e.g., that if we are mining on months, $Maxw \leqq 12$, if we are mining on weeks, $Maxw \leqq 53$, and if we are mining on minutes, $Maxw \leqq 60$. We take for granted that a time-dimension is constructed such that this is possible since that is the implementation "best practice" of TARGIT. The reason for limiting *Maxw* not to exceed the timeframe of a parent is, that we only want to find the sentinel for warning period, *w*, and not for warning periods $= w + (\text{parent timeframe} \times n) | n \in \mathbb{N}$. This would be the case if a sentinel is cyclic over time, and we did not limit *Maxw*. For cyclic sentinels we prefer the shortest warning period, w, since we attribute greater quality to sentinels based on the "latest intelligence from the operational environment", as opposed to a long-term prediction pattern. In addition, a shorter warning period means that more cycles in the OODA loop are possible based on the sentinel, and therefore an organization will be able to react and adapt faster to its environment.

The remaining parameters for o, $MaxSource$, wp, $RPGpareto$, σ, γ, β, and X are silently applied by the system for most users. It is possible for expert users to change a registration file to modify these. However, the "few click" ambitions in the TARGIT BI Suite suggests that the system makes a "best bet" choice on behalf of casual users. The parameters have been selected based on experiments with sentinel mining on real world data from TARGIT A/S, NASDAQ OMX stock exchange, and the governmental institute Danish Statistics.

Step 2: In this step, the *SentHiRPG* function is called with three different values of α. The three set of sentinels mined competes with the maximum *Score* found in the set, and the set that "wins" is kept in memory. The reasons for testing specifically these three values of α is rooted in the same real world experiences as stated under Step 1.

Step 3: This step returns the X best sentinels found along with the length of the warning period, w, and the value of α. The sentinels found passes through the Local Language Translation Layer in order to be localized for the user. In practice this means that all the measures that constitute the sentinels are translated to the user's language. The reason for having X is to limit the load on the visual interface of the client dialogue. It was found that a stress load of sentinels to the client would overload the visual control, and since several hundreds (or thousands) of rules would not contribute meaningfully to the users overview anyways, a threshold was set to 200. Based on real world experiences this threshold is rarely exceeded, and thus most often all sentinels are returned to the client.

Computational Complexity of Sentinel Mining: When we examine the individual steps in the Sentinel Mining function, we note that the scan of the individual cubes from *CubeSet* in Step 1 can be performed in time $\mathcal{O}(n)$, where *n* is the number of periods in *T* for which we have data. This is justified by the fact, that each individual cube scan can be performed in $\mathcal{O}(n)$ under the assumption that the cube is aggregated on *T*, and that each aggregated cell can be accessed through a hash-based look-up [22]. Each of these individual cube scans produces a unique part of *C* that can be mapped directly (with no index lookup) using *T* at a small, constant performance overhead. The number of individual cube scans in *CubeSet* is also a constant, and this constant is multiplied by *n* since we need to go through the entire dataset. Since both these constants can be disregard, Step 1 can be performed in time $\mathcal{O}(n)$. In Step 3, we note that the scan of the input of size n, (Step 2 output) produces an output that is at most as large as the input, so this can be performed in time $\mathcal{O}(n')$ and the computational complexity will be dominated by the size of the input. However, since $n' \leqq n$, Step 3 can thus be disregarded.

According to Chapter 3, *SentHiRPG* has a computational complexity of $\mathcal{O}(n + c \times p(q)^l \times k^l \times m)$ where *n* is the size of *C*, *c* is the number of unique facts in *C* (which in this implementation is the size of *T*), *p* is the percentage of remaining source measures after RPG optimization expressed as a function of *q*, where *q* is *RPGpareto*, *l* is *MaxSource*, *k* is the number of source measures, and *m* is *Maxw*. Repeating *SentHiRPG* three times in Step 2 does not change this computational complexity since the multiplying constant can be disregarded.

Since Step 1's computational complexity of $\mathcal{O}(n)$ is already included in the *SentHiRPG* computational complexity, we say that the entire implementation of Sentinel Mining in the TARGIT BI Suite has a computational complexity of $\mathcal{O}(n + c \times p(85\%)^3 \times k^3 \times m)$ with the variables defined above, with the exception of *q* and *l* that have been replaced by the constants for *RPGpareto* and *MaxSource* in alignment with the implementation of Step 1. The size of $n = c \times (k + 2)$, since *C* is a table with $|T|$ rows and $k+2$ columns (the number of source measures plus one column for target measure and one column for the dimension *T*). In meaningful sentinel mining we have that $2 << m$, and we can thus disregard the constant. With this we can simplify the computational complexity to $\mathcal{O}(c \times k \times (1 + \times p(85\%)^3 \times k^2 \times m)$, and eliminating the constant we end with a computational complexity for our implementation of $\mathcal{O}(c \times p(85\%)^3 \times k^3 \times m)$.

With this notion, we will expect linear behavior when scaling the size of *T*. The length of the period, *Maxw* (m), in which we fit the warning period is also expected to scale linearly. When scaling the number of source measures (k) we will expect the running time to rise cubically without the RPG optimization. However, since the effect of the RPG optimization $(p(85\%))$ is also cubic we will expect this to be an efficient countermeasure to the impact of scaling the number of source measures. This analysis is verified in Section 5.5.

Using sentinels after discovery: Upon completion of the Sentinel Mining process, the sentinels found will be listed in a user dialogue as shown in Figure 5.1(b). The dialogue allows the user to select any number of the sentinels found, and to schedule these for notification at a given frequency. The frequency defaults to the same period as the hierarchical level of *T*, since it rarely makes sense to test for the existence of the premise for a sentinel on a shorter period than it was mined for. Upon scheduling, the sentinel is basically a traditional *agent*, that tests with the frequency set for the existence of the premise for the sentinel in the most recent period of *T* at the level for which it was mined. If the premise for the sentinel is found, the user is notified by email, directly on his computer desktop, or iPhone with a message as shown in Section 5.2.

As mentioned in Section 5.2, the confidence of a sentinel is fluid as time progresses since the user's actions will interfere with the "predictive power" of the sen-

tinel. In fact, all the quality measures are variable as time progresses. However, we choose not to reconsider *Balance* and *Score* since the causality and ranking was only important when the user had to identify one or more relevant sentinels among others. Now that the user has deemed a given sentinel relevant, there is no reason to reconsider the relevancy quantitatively. There is of course the option that a user can unschedule a sentinel manually at any time if he does not find it relevant anymore.

The sentinel prediction and confidence is shown in a notification to the user, if confidence meets the threshold. If the confidence of a sentinel tested falls short of the threshold, the user is notified that the sentinel is recommended to be *retired* as described in Section 5.2. The retirement of a sentinel from active notification concludes the life-cycle of a sentinel.

The cost of checking for the premises for all sentinels is linear in the number of sentinels times the number of source measures per sentinel, since each check requires reading two cells for each included source measure. We recall that the cube (per recommendation) is aggregated on T, and thus each aggregated cell can be accessed through a hash-based look-up [22]. Thus, the cost of "sentinel check" is much lower than the cost of mining sentinels, and can typically be performed in a few seconds. This assumption is verified in Section 5.5.

5.4 Market Experiences

Sentinels was launched as a new feature in the TARGIT BI Suite version 2K9 which was released to the market in April 2009. This first implementation was based on an enhanced version of the algorithm described in Chapter 2. From version 2K9 SR1 (service release 1), released in November 2009, and forward, the implementation of the algorithm from Chapter 3 is done as described in this chapter. Since most sentinels are likely to be based on business data where a weekly or monthly warning period is desired, time has simply not permitted other success stories with sentinel usage than the internal experiences in TARGIT A/S. However, given TARGIT's ten years of experience with marketing software through an indirect sales model, we know that the partners that sell and implement the TARGIT BI Suite are a trustworthy source for fair initial market feedback. This feedback has been encouraging so far, in 2009 partners that represented a market footprint of 1,936 customers with more than 124,000 users rated sentinels the most interesting and promising feature.

In addition to TARGIT's own surveys, leading industry analyst Gartner has recently accepted TARGIT into the so-called Magic Quadrant for Business Intelligence Platforms in 2010 [54]. The sentinel technology is specifically listed by Gartner as one of the unique key strengths of the TARGIT:

"The introduction of an innovative alerting solution, called Sentinels (essentially prediction-based rules), enables an end user to react quickly to alerts for certain indicators. Through the combination with Targit's desktop alerts, a user gets an early-warning notification when a predefined rule has been violated and the user can proactively take corrective measures. This capability adds to Targit's attractiveness for end users."

Even though the business impact of sentinels is expected, but not yet to be seen in the large scale, we have learnt a few things about the user interaction with the sentinel mining implementation. With regards to the dialogue where the sentinels found are presented, Figure 5.1(b), it has been desired by users to see the *Score* for each sentinel as a number. In the current implementation only confidence is presented as a number, whereas *Score* is discretely presented as a sort order where the user knows that the best sentinels are on top of the list. In addition, it has been desired by users to have easy access to a visualization of the relationship between the source measures and target measure. This can be done by allowing the user to see the measures that constitute the sentinel presented as bar-charts alongside with the changes highlighted (color-coded). This should of course be available with a single click with reference to Section 5.2.

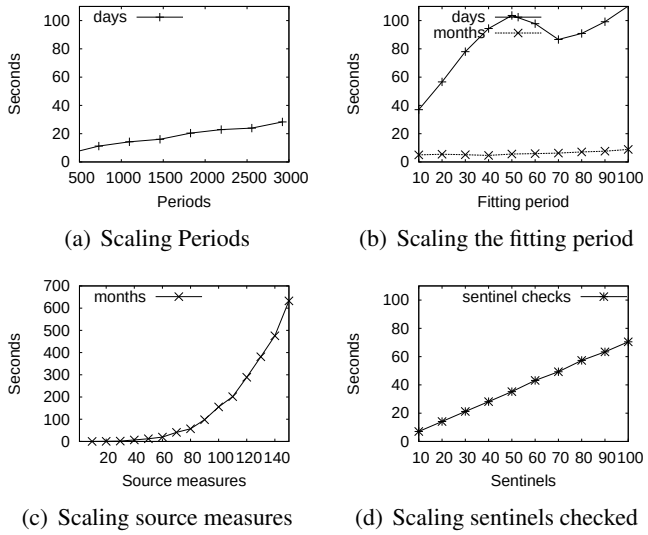
5.5 Experiments

Setup: We run our experiments directly on the data of the live TARGIT data warehouse described in Section 5.2. Our setup is implemented across two physical servers:

1. a Quad Core Intel Xeon X3220 2.4 GHz with 8 GB RAM, 2 x 73 GB SAS harddrives in RAID 1, running MS SQL Server 2008 cubes on a Windows 2008 64-bit operating systems.

2. an Intel Core2 Quad CPU (Q6600) 2.40GHz PC with 4GB RAM and 2 500GB disks (7,200 RPM) running the TARGIT ANTserver on a 64Bit version of Windows 2003 Server R2, Service Pack 2.

We run on version 2K10 of the TARGIT BI Suite (currently in development) that is expected to launch in April 2010. A client is accessing the TARGIT ANTserver, and this client has the following specification: Intel Core2 Dual CPU (6400) 2.13GHz PC with 2GB RAM and 1 250GB disks (7,200 RPM) running a 32Bit version Windows 7. However, it should be noted that the client did not do any processing of the data, it simply relayed a "sentinel mining" or "sentinel check" query to the TARGIT ANTserver on data warehouse server #2 and waited for a reply. Upon reply, the time



(a) Scaling Periods (b) Scaling the fitting period

(c) Scaling source measures (d) Scaling sentinels checked

Figure 5.3: Sentinel Mining Experiments

consumed was recorded. The experiments shown in Figure 5.3 and 5.3(d) represent the average of 5 runs each, and the sequence of all the experiments is randomized. To run queries on 10, 20, 30, ..., 140 source measures, we produce descendants of the full data warehouse by selecting a given number of source measures randomly. Please note that the time-scale on Figure 5.3(c) is different from all other figures.

In Section 5.3 we describe how we reduce the number of measures by considering only "base" measures in the mining process. In addition, we naturally only mine the measures that have a relationship (a shared time-dimension). Whenever we refer to a number of measures in the following experiments, we refer to the number of related base measures, e.g., in the TARGIT case there are 250 measures in total, but the largest possible number of related base measures is 154. The TARGIT data warehouse contains data on all related base measures from January 2002 and forward, meaning that we have 8 complete years of data, 2002 to 2009, to run our experiments on.

Scaling Periods: In Figure 5.3(a) we scale the size of the input, i.e., the number of periods over which we search for sentinels, on 50 measures (49 source measures and 1 target measure), and with a fitting of the warning period over 7 days (1 week, the duration of the parent level of the time dimension). To get the largest possible dataset, we run the sentinel mining on the "days" level of the time hierarchy which gives us a maximum of 2,922 periods for 8 years. We start by mining on data from

2009 which accounts for 365 days. From this point, we add one year at the time to increase the number of periods with either 365 or 366 days depending on the year. The results are presented in Figure 5.3(a), and we note that the sentinel search scales linearly from 365 to 2,922 periods as expected based on our assessment of the computational complexity (when the other factors are kept constant). We should also note that 2,922 periods in this context is a *huge* dataset, since users would normally be operating on higher aggregated levels such as weeks or months. However, even at this finest granularity in the TARGIT data warehouse, the linear scalability seems to more than adequate to facilitate real world usage on a broad user scale, since the mining on even this finest granularity can be conducted for the individual user in less than 30 seconds (we recall that the cube data is pre-aggregated at the period(s) and level(s)).

Scaling the fitting period: In order to scale the time frame, in which we fit the warning period, as seen in Figure 5.3(b), we needed to tweak the default settings in the TARGIT ANTserver to omit the built in prevention method for finding the same cyclic sentinel multiple times as described in Section 5.3. In this experiment, we mine sentinels on 50 measures (49 source measures and 1 target measure) on "days" and "months" granularity respectively, and we scale the "fitting period" from 10 to 100 periods. Based on our assessment of computational complexity, we would expect both levels on the time dimension to scale linearly (when the other factors are kept constant). We do indeed observe a linear behavior for "days" and "months" when the fitting period is within the default settings, i.e., "days" will default to a fitting period of either 7 or 31 periods (days), and "months" would default to a fitting period of 12 periods (months). However, when scaling beyond the default settings only the scaling on the "months" level displayed a linear behavior. The "days" level seemed linear in behavior from 10 to 50 periods, but then dropped in time consumption from 50 to 70 periods, from where it again displayed linear behavior. The reason is, that another local maximum for *Score* exists close to a warning period of 70 which has better score than a local maximum between 1 and 10. This means that our hill climbing part of the SentHiRPG algorithm will have a tendency to be attracted towards the longer warning periods once "fitting period" $\geqq 60$. Such a signature suggests that the sentinels mined on "days" are cyclic on a bi-monthly basis. However, as explained in Section 5.3, we maintain that a shorter warning period is usually more relevant and actionable for a user than a longer period.

Another interesting finding in Figure 5.3(b) is that, although the mining on "days" is linear in two period-sequences, it seems to be much more costly than mining sentinels on "months" when scaling the "fitting period". When investigating this, we found that the sentinel mining only finds two sentinel on the "days" level that actually meets the thresholds, meaning that all the time is used to process a lot of very

similar, but very poor sentinels. At the moment, the RPG optimization benefits from the existence of influential measures in the input data, but if a lot of poor measures, and no good measures, are input then the effect of the RPG optimization is low. Effectively, this means that we spend *more* time finding *less* and *poorer* sentinels. We should note that even though the sentinel mining is not used on an optimal dimension level from neither a performance nor quality perspective, the performance of the system is still fair, since even the longest running query takes less than two minutes. This is more than adequate for all users, since the user does not have to be online while conducting the sentinel mining. In this context, we also note than when running on the "month", which produces more useful sentinels from a business perspective, the longest running mining process takes less than 9 seconds.

Scaling source measures: In Figure 5.3(c) we scale the number of source measures from 10 to 150 on a realistic sentinel mining process at the 'months" level of the time hierarchy. During this process the warning period is fitted over 1 year. We notice that increasing the number of source measures is expected to have a cubic impact based on our assessment of the computational complexity, and we have verified in the statistical tool R (`www.r-project.org`) that the curve in Figure 5.3(c) is indeed cubic. Although the curve seems steep, we should note that the impact of scaling the source measures would be more than seven times higher without the effect of the RPG optimization (RPG reduces the workload of combining source measures to 14% on real data as described in Chapter 3). We attribute the efficiency of the RPG optimization on real data to the existence of a power law in the data, where a few source measures have strong relationships with the target measure whereas a larger number of source measures has little or no relationship with it. From a user perspective the performance is adequate, since the longest sentinel mining process possible in the TARGIT data warehouse at this level took 10.5 minutes, and as we recall, the user does not need to be online while mining. Moreover, the wait is worthwhile since the quality of the sentinels found will benefit the user with early warnings for months to come. We recall that once mined, the sentinel can be scheduled and "checked" (whether to warn or not) with a given frequency, and the time consumption of a sentinel check is less than a second in a real, operational data warehouse.

Scaling sentinels checked: In Figure 5.3(d) we scale the number of sentinels for which we check for the existence of their respective premises. We note a complete linear scalability when scaling to a realistic number of sentinels for a reasonably large "sentinel swarm" (see Section 5.2). This is consistent with what we would expect based on our assessment of computational complexity of such a process (see Section 5.3). We recall that our sentinel mining returned 26 possible sentinels for *Revenue* in Figure 5.1(b), and thus 100 sentinels is roughly equal to scheduling four

sentinel swarms to guard four critical measures. The cost of checking the entire number of sentinels requires just about one minute of processing (precisely 1 minute and 10.5 seconds), which means that a monthly follow-up on a sentinel swarm is realistic for a large group of users.

Experiments Summary: In summary we have demonstrated that the sentinel mining in the TARGIT BI Suite (with default settings) scales linear in number of periods and length of fitting period, and it scales cubically in number of source measures. We confirmed that the current settings in the TARGIT BI Suite are adequate to prevent users from mining cyclic sentinels, and that these settings will also prevent users from spending excessive time and system resources in doing so. Most importantly, we have demonstrated that both mining and usage of sentinels is feasible with good performance for the typical users on a real, operational data warehouse.

5.6 Related Work

The industry analyst Gartner is a well-known authority in the business intelligence industry, and the Magic Quadrant analysis is by many considered the most influential description of the top international vendors. The Magic Quadrant analysis for 2010 [54] categorizes 15 vendors as either "market leaders" (7 companies), "challengers" (3 companies), or "niche players" (5 companies). The companies identified as "market leaders" are: IBM, Oracle, Microsoft, SAS, SAP, Information Builders, and MicroStrategy. Gartner categorizes features such as sentinels as "predictive modeling and data mining", and all "market leaders" have features within this category. However, only SAS seems to be significantly differentiated from the other "market leaders" in this category with a more comprehensive predictive offering since the company originated from forecasting and predictive modeling, whereas the other companies started as DBMS providers or as providers of reporting centric solutions. Out of all features offered by the "market leaders" [8, 28, 34, 35, 47, 56], the algorithms that to some extent resemble the functionality of sentinels are *association rules*, *sequential patterns*, and *regression techniques* (see comparison below). This also holds for the "challenger" Tibco, which is the only other noteworthy company with "predictive modeling and data mining" features. As explained in detail below, these competing techniques are distinctly different from the sentinel technology implemented by TARGIT. Moreover, TARGIT is a new entrant in the "niche player" category of the Magic Quadrant, and with reference to Gartner's statement in Section 5.4, the sentinels of TARGIT are also perceived as a unique feature in the Magic Quadrant analysis. In general, TARGIT is seen by Gartner as a "niche player" with a strong foothold in the mid-market, and with a unique position in its approach to BI usability. This is in line with the "few clicks" approach explained in Section 5.2, and

therefore the entire concept of sentinels is unique from a market perspective.

From a scientific perspective, the idea that some actions or incidents are interlinked has been well explored in *association rules* [2]. The traditional case study of association rules has been basket-type association rules, and significant effort has been put into optimizing the original Apriori algorithm [3, 11]. In general, association rule mining seeks to find co-occurrence patterns within *absolute data values*, whereas our solution works on the *relative changes in data*. In addition, association rule mining typically works on *categorical data*, i.e., dimension values, whereas our solution works on *numerical data* such as measure values. *Sequential pattern mining* adds to the complexity of association rules by introducing a sequence in which actions or incidents take place [5], thus new optimization approaches have emerged [24, 48, 49, 57]. Sequential pattern mining allows a time period to pass between the premise and the consequent in the rule, but it remains focused on co-occurrence patterns within absolute data values for categorical data. Furthermore, our solution generates rules at the *schema level*, as opposed to the *data level*, using a contradiction elimination process. The schema-level property allows us to generate fewer, more general, rules that cannot be found with neither association rules nor sequential pattern mining. In Chapter 2 we demonstrate why sequential pattern mining does not find any meaningful rules compared to simple sentinel rule discovery, referred to as "baseline" in Section 5.5. Compared to generalized sentinel rules, the data level nature of sequential pattern mining means that it can neither identify bi-directional rules that represent the "strongest" causal relationships, nor can it qualify such relationships with a balance assessment against a threshold. The schema level nature of generalized sentinel rules gives rise to the table of combinations (TC) and the reduced pattern growth (RPG) optimization, and such optimization can therefore not be offered by sequential pattern mining or other known optimizations for simpler "market basket"-type data such as [11]. In addition to the TC and RPG optimizations, the auto-fitting of the warning period, and the ability to combine source measures into better sentinel rules, adds to the distance from our solution to sequential patterns.

Gradual rule mining [7, 10, 27, 31] is a process much like association rules, where the categorical data are created by mapping numerical data to fuzzy partitions, and thus this technique works on numerical data similar to sentinels. However, similar to association rules and sequential patterns, gradual rule mining does not have the schema level property of sentinels that allows sentinel mining to create the strong bi-directional rules.

Other approaches to interpreting the behavior of data sequences are various regression [4] and correlation [26,60] techniques which attempt to describe a functional relationship between one measure and another. In a multi-dimensional database such regression techniques (Bellwether Analysis) can be used on historical sales data to identify a leading indicator that can predict the global sales of a new product, for

which only a short period of data is available [13, 14]. However, similar to gradual rules, these techniques are also concerned with the absolute values of a measure, as opposed to sentinels that are based on changes in the measure values. With regards to the output, sentinels are more *specific "micro-predictions"*, i.e., strong rules that hold for a subset of the data and stimulate specific actions, and are thus complementary to these techniques. Sentinels are therefore useful for detecting incidents and generating warnings whenever changes (that would otherwise go unnoticed) in a relevant source measure occur (Chapter 2).

The CALM theory [36] describes how BI technologies in general, and sentinels in particular, can be integrated in an OODA loop (Appendix A). The concept of simple sentinel rules has been described in Chapter 2, and was significantly extended into generalized sentinel rules (Chapter 3), that allow multiple source measures to be combined into better rules through a quality assessment that can also auto-fit the best warning period. The user interaction in sentinel mining in the TARGIT BI Suite is described in Chapter 4. In comparison, this chapter describes an industrial implementation of the sentinel concept from a user and data perspective. In addition, we provide detailed description of the implementation of sentinels in a standardized software package, and subject this software to several experiments on a real, operational data warehouse.

5.7 Conclusion and Future Work

Motivated by the need for business users to make fast decisions, we showed how users without any knowledge of databases and data mining were able to search, schedule and receive warnings from sentinels. We described an implementation where little training is needed for a user to benefit from the sentinel technology. We demonstrated sentinel mining from both a user and a data perspective, and we specifically demonstrated how to score the best sentinels. We described in detail how an algorithm for sentinel mining was implemented in the server layer of the TARGIT BI Suite, and how the user interaction with this layer takes place. We conducted several experiments and showed that both mining and usage of sentinels is feasible with good performance for the typical users on a real, operational data warehouse. Furthermore, we identified the state of the art technologies in both industry and science, and we showed that these technologies are distinctively different from sentinels, and we pointed out where sentinels are useful when other technologies are not. In summary, we demonstrated this implementation of sentinel technology to be effective, useful, and unique.

For future work we would like to incorporate the experiences from the market (see Section 5.4), where an ability to visualize sentinels was desired in order to give users an understanding of what is going on behind the scenes. Such a visualization

will most likely assist in persuading more users to trust and thus benefit from sentinels. In addition, the observations in Section 5.5 gave us an idea to remove the source measures that have less influence than the *SentSupp* threshold prior to running the RPG. This modification is expected to restore the effect of the RPG optimization even on data without strong sentinels. Finally, we would like to induce more flexibility by introducing intervals to replace the fixed warning period and the offset.

Chapter 6

Efficient Sentinel Mining Using Bitmaps on Modern Processors

This chapter proposes a highly efficient bitmap-based approach for discovery of so-called *sentinels*. Sentinels represent *schema level* relationships between *changes over time* in certain measures in a multi-dimensional data cube. Sentinels notify users based on previous observations, e.g., that revenue might drop within two months if an increase in customer problems combined with a decrease in website traffic is observed. We significantly extend prior work by representing the sentinel mining problem by bitmap operations, using *bitmapped encoding* of so-called *indication streams*. We present a very efficient algorithm, *SentBit*, that is 2–3 orders of magnitude faster than the state of the art, which can utilize CPU specific instructions and the multi-core architectures available on modern processors. The *SentBit* algorithm scales efficiently to very large datasets, which is verified by extensive experiments on both real and synthetic data.

6.1 Introduction

The Computer Aided Leadership and Management (CALM) concept copes with the challenges facing managers that operate in a world of chaos due to the globalization of commerce and connectivity [36]; in this chaotic world, the ability to continuously *react* is far more crucial for success than the ability to *long-term forecast*. The idea in CALM is to take the Observation-Orientation-Decision-Action (OODA) loop (originally pioneered by "Top Gun" fighter pilot John Boyd in the 1950s), and integrate business intelligence (BI) technologies to drastically increase the speed with which

a user in an organization cycles through the OODA loop. One way to improve the speed from observation to action is to expand the "time-horizon" by providing the user of a BI system with warnings based on "micro-predictions" of changes to an important measure, often called a Key Performance Indicator (KPI). A *sentinel* is a causal relationship where changes in one or *more source measures*, are followed by changes to a *target measure* (typically a KPI), within a given time period, referred to as the *warning period*. We attribute higher quality to *bi-directional* sentinels that can predict changes in both directions, since such a relationship intuitively is less likely to be coincidental (see Section 6.2). An example of a sentinel for a company could be: "IF Number of Customer Problems go up and Website Traffic goes down THEN Revenue goes down within two months AND IF Number of Customer Problems go down and Website Traffic goes up THEN Revenue goes up within two months". Such a rule will allow a BI system to notify a user to take corrective action once there is an occurrence of, e.g., "Customer Problems go up and Website Traffic goes down", since he knows, based on the "micro-prediction" of the rule, that Revenue, with the probability stated by the rule's confidence, will go down in two months if no action is taken.

Our contributions are as follows. First, we show how to represent the sentinel mining problem by bitmap operations, using *bitmapped encoding* of so-called *indication streams*. Second, we present a very efficient algorithm, *SentBit*, that is 2–3 orders of magnitude faster than the state of the art. *SentBit* does not use approximation, and thus provides exact results unlike prior art. Third, we provide a number of optimizations utilizing CPU specific instructions and the multi-core architectures available on modern processors. Fourth, we present experiments demonstrating that *SentBit* scales efficiently to very large datasets, and that sentinels are only found if the data contains statistically significant relationships.

Compared to prior art, sentinels are mined on the measures and dimensions of multiple cubes in an OLAP database, as opposed to the "flat file" formats used by most traditional data mining methods. Sentinels find rules that would be impossible to detect using traditional techniques, since sentinels operate on *data changes* at the *schema level* as opposed to absolute *data values* at the *data level* such as association rules [2] and sequential patterns typically do [5]. In Chapter 2 we specifically provide a concrete, realistic example where nothing useful is found using these techniques, while sentinel mining *do* find meaningful rules. The nature of changes at the schema level gives rise to bitmapped encoding (Section 6.4) of indication streams (Section 6.3), which is the prerequisite of bitmapped sentinel mining, and therefore prior art will not be able to find the same rules. In addition, the auto-fitting of the warning period, and the ability to combine source measures into better sentinel rules, adds to the distance between our solution and the results and optimizations offered in prior art such as [3, 11, 24, 46, 48, 49, 53, 57, 59].

Gradual rule mining [7,10,27,31] is a process much like association rules, where the categorical data are created by mapping numerical data to fuzzy partitions, and thus this technique works on numerical data similar to sentinels. However, similar to association rules and sequential patterns, gradual rule mining does not have the schema level property of sentinels that allows sentinel mining to create the strong bi-directional rules.

Other approaches to interpreting the behavior of data sequences are various regression [4] and correlation [26,60] techniques which attempt to describe a functional relationship between one measure and another. In a multi-dimensional database such regression techniques (Bellwether Analysis) can be used on historical sales data to identify a leading indicator that can predict the global sales of a new product, for which only a short period of data is available [13, 14]. However, similar to gradual rules, these techniques are also concerned with the absolute values of a measure, as opposed to sentinels that are based on changes in the measure values. With regards to the output, sentinels are more *specific "micro-predictions"*, i.e., strong rules that hold for a subset of the data and stimulate specific actions, and are thus complementary to these techniques. Sentinels are therefore useful for detecting incidents and generating warnings whenever changes (that would otherwise go unnoticed) in a relevant source measure occur (Chapter 2).

With regards to parallelization of our bitmapped sentinel mining algorithms, prior art in other aspects of data warehousing have also applied parallelization in order to deal with the huge volumes of data that resides in these systems [17,18]. In addition, multi-core parallelism has been applied to gradual rule mining [31].

Sentinel mining from a user and an industry perspective has been described in Chapters 4 and 5, and the underlying algorithms have been described in Chapters 2 and 3. In comparison, this chapter provide an algorithm using bitmaps that is 2–3 orders of magnitude faster.

The remainder of the chapter is structured as follows: The next section intuitively presents the concept of sentinels, Section 6.3 presents the formal definition, Section 6.4 presents the prerequisites for bitmapped sentinel mining, Section 6.5 presents the new *SentBit* algorithm, and Section 6.6 presents the optimization and implementation of *SentBit*, Section 6.7 presents experimental results, and Section 6.8 presents our conclusions and proposals for future work.

6.2 The Sentinel Concept

Table 6.1 is a data example for a company, where two subsets have been extracted from a database. We have assigned short names to the measures as follows: *PeoInv* = the number of people involved in the decision process for customer projects, *UniRev* = the revenue of training activities, *WHts* = the number of human hits on the com-

(a) Source

Month	*PeoInv*	*UniRev*	*WHts*	*Ind(PeoInv)*	*Ind(UniRev)*	*Ind(WHts)*
2009-Jan	1	115	1320			
2009-Feb	2	115	1310	▲		
2009-Mar	2	100	1490		▼	▲
2009-Apr	3	90	987	▲	▼	▼
2009-May	2	363	888	▼	▲	▼
2009-Jun	3	310	1147	▲	▼	▲
2009-Jul	2	440	1003	▼	▲	▼
2009-Aug	4	297	1150	▲	▼	▲
2009-Sep	5	260	993	▲	▼	▼
2009-Oct	6	230	1110	▲	▼	▲
2009-Nov	4	294	1200	▼	▲	
2009-Dec	5	264	1420	▲	▼	▲
2010-Jan	6	230	1350	▲	▼	
2010-Feb	4	270	1380	▼	▲	
2010-Mar	3	353	1530	▼	▲	▲
2010-Apr	2	400	1310	▼	▲	▼

(b) Target

Month	*Rev*	*Ind(Rev)*
2009-Apr	900	
2009-May	1001	▲
2009-Jun	1200	▲
2009-Jul	750	▼
2009-Aug	1001	▲
2009-Sep	1100	
2009-Oct	1250	▲
2009-Nov	970	▼
2009-Dec	850	▼
2010-Jan	720	▼
2010-Feb	1250	▲
2010-Mar	930	▼
2010-Apr	800	▼
2010-May	1100	▲
2010-Jun	1400	▲
2010-Jul	1600	▲

Table 6.1: The relationship between two source measures and a target measure

pany's website, and *Rev* = revenue for the entire company. The source measures, *PeoInv*, *UniRev*, and *WHts*, in Table 6.1(a) have been extracted for January 2009 to April 2010. The target measure, *Rev*, in Table 6.1(b) has been extracted for April 2009 to July 2010; a similar period in length starting three months later. We refer to these three months as the *Warning Period*. We have calculated the cases where a measure changes 10% or more, either up (▲) or down (▼), from one month to another. We refer to each change to a measure over time as an indication, *Ind*.

As seen in the 16 rows in Table 6.1, the measures *PeoInv* and *UniRev* tend to change in a combined pattern such that when *PeoInv* goes up, *UniRev* goes down, and vice versa. This source measure pattern is observed 13 times, out of 15 possible. If we combine this pattern with the subsequent changes to *Rev* three months later, we see that *Rev* changes in the same direction as *UniRev* in 12 out of 13 possible times (denoted by $\#ChangesToSource = 13$). Another observation is that the relationship between *Rev* and the combination of *PeoInv* and *UniRev* goes in both directions, which is a property we refer to as *bi-directionality*. Intuitively, one can say that if a relationship is bi-directional, then there is a greater chance that the relationship is causal, as opposed to a uni-directional relationship where a pattern is observed for measure changes in one direction only. Consider a case where revenue and staff costs increase over a period of time. This yields the uni-directional relationship that an increase in revenue leads to an increase in staff costs the following month; in this case a decrease in revenue will not necessarily lead to a decrease in staff costs since these costs tend to be more fixed. Therefore, bi-directional relationships are more desirable. It is also noteworthy that *Rev* changes 6 times up (denoted by $A = 6$) and 6 times down (denoted by $B = 6$) in combination with *PeoInv* and

UniRev since this "balance" again adds to the likeliness that the relationship is indeed causal. In summary we can say that a sentinel exists in Table 6.1 where changes in *PeoInv* and *UniRev* is able to warn three months ahead about changes to *Rev* with a *Confidence* of 92% (12 out of 13 times), defined as $Confidence = \frac{|A+B|}{\#ChangesToSource}$. $Balance = \frac{4\times|A|\times|B|}{(|A|+|B|)^2}$ is a measure for the degree to which a sentinel is balanced, and in this case the sentinel is perfectly balanced, meaning that $Balance = 1$. We note that the bi-directional quality that can be achieved by assessing *Balance*, is impossible to achieve for sequential patterns since they can only represent one direction of changes in each pattern.

In addition to the combined relationship of the source measures, we can also observe "simple" sentinels (Chapter 2) with only one source and one target measure in Table 6.1. However, the *inverted* relationship between *PeoInv* and *Rev*, as well as the relationship between *UniRev* and *Rev*, each have one occurrence (the first two changes) where *Rev* changes in the opposite direction of what we would expect from all other changes. To assess the prediction ability for such sentinels we must first eliminate its *internal contradictions*. This is done by deducting the number of times *Rev* changes in the "unexpected" direction from the number of times *Rev* changes in the "expected" direction. This means that both source measures change 14 times, whereas the target measure after elimination changes only 11 times $(12 - 1)$. Therefore the simple sentinels have a poorer *Confidence* of 79% ($\frac{5+6}{14}$) and are slightly less balanced ($Balance = \frac{4\times|5|\times|6|}{(|5|+|6|)^2} = 0.99$) compared to the sentinel where the source measures were combined. On the other hand, simpler sentinels with fewer source measures have the advantage of being more general than very specific, potentially *overfitted*, sentinels with many source measures, and therefore the simplicity of a sentinel is also important.

6.3 Formal Definition

Let C be a multi-dimensional *cube* containing a set of *facts*, $C = \{(d_1, d_2, ..., d_n, m_1, m_2, ..., m_p)\}$. The dimension values, $d_1, d_2, ..., d_n$, belong to the *dimensions* $D_1, D_2, ..., D_n$, and we refer to the "dimension part" of a fact, $(d_1, d_2, ..., d_n)$, as a *cell*. We say that a cell belongs to C, denoted by $(d_1, d_2, ..., d_n) \in C$, when a fact $(d_1, d_2, ..., d_n, m_1, m_2, ..., m_p) \in C$ exists. We say that a *measure value*, m_i, is the result of a partial function, $M_i : D_1 \times D_2 \times ... \times D_n \hookrightarrow \Re$, denoted by, $M_i(d_1, d_2, ..., d_n) = m_i$, if $(d_1, d_2, ..., d_n) \in C$ and $1 \leqq i \leqq p$. We refer to M_i as a *measure*. We assume, without loss of generality, that there is only one time dimension, T, in C, and that $T = D_1$, and subsequently $t = d_1$. In addition, we assume that measures $M_1, ..., M_{p-1}$ are source measures, and that measure M_p is the target measure. An *indication*, *Ind*, tells us whether a measure, M_i, changes

by a factor of at least α over a period, o. We define $Ind(M_i, t, o, d_2, d_3, ..., d_n)$ when $\{(t, d_2, d_3, ..., d_n), (t + o, d_2, d_3, ..., d_n)\} \subseteq C$ as shown in Formula 6.1.

$$Ind(M_i, t, o, d_2, d_3, ..., d_n) = \begin{cases} \blacktriangle \text{ if } \frac{M_i(t+o, d_2, d_3, ..., d_n) - M_i(t, d_2, d_3, ..., d_n)}{M_i(t, d_2, d_3, ..., d_n)} \geqq \alpha \\ \blacktriangledown \text{ if } \frac{M_i(t+o, d_2, d_3, ..., d_n) - M_i(t, d_2, d_3, ..., d_n)}{M_i(t, d_2, d_3, ..., d_n)} \leqq -\alpha \end{cases} \tag{6.1}$$

We refer to ▲ as a *positive* indication and to ▼ as a *negative* indication. We define a wildcard, ?, that can be either ▲ or ▼. In addition, we define the *complement* of an indication as follows: $\overline{\blacktriangle} = \blacktriangledown$ and $\overline{\blacktriangledown} = \blacktriangle$. Furthermore, we define the inverted measure, $\overline{M_i(x)} = -M_i(x)$, thus all indications $Ind(\overline{M_i}, t, o, d_2, d_3, ..., d_n) = \overline{Ind(M_i, t, o, d_2, d_3, ..., d_n)}$.

We define an *indication sequence*, *IndSeq*, for a measure, $M_i \in \{M_1, ..., M_p, \overline{M_1}, ..., \overline{M_p}\}$, as shown in Formula 6.2. Intuitively, an *IndSeq* captures the indications over time for each dimension combination, $(t, d_2, d_3, ..., d_n)$. In the following we use set notation for convenience, but implicitly assume that the *order* of the sequence is given by the tuple $(t, d_2, d_3, ..., d_n)$.

$$\begin{aligned} &IndSeq(M_i, C, o, w) = \\ &\quad \{(t, d_2, d_3, ..., d_n, Ind(M_i, t + w, o, d_2, d_3, ..., d_n)) \mid \\ &\qquad \{(t, d_2, d_3, ..., d_n), (t + w, d_2, d_3, ..., d_n), \\ &\qquad (t + o + w, d_2, d_3, ..., d_n)\} \subseteq C \wedge \\ &\qquad Ind(M_i, t + w, o, d_2, d_3, ..., d_n) \in \{\blacktriangle, \blacktriangledown\}\} \end{aligned} \tag{6.2}$$

We define a join of two or more indication sequences, *IndJoin*, for a set of measures, $\{S_1, ..., S_m\} \subseteq \{M_1, ..., M_p, \overline{M_1}, ..., \overline{M_p}\}$ as shown in Formula 6.3. Formula 6.3 allows optional differences in time, $w_1, ..., w_m \in \mathbb{N}_0$, between the indication sequences joined, referred to as warning periods. Intuitively, an indication join is an *m-way semi-join* where we take the left indication sequence and filter out all indications that do not fit with the other sequences. For source measures it is typical to combine measures for the same time instance, and for target measures for a later time instance (default $w_x = 0$ for all $w_x | 1 \leqq x \leqq m$). In addition, we define an optional filter, $F \in \{\blacktriangle, \blacktriangledown, ?\}$, that allows the resulting indication sequence to consist of indications in one direction only (default $F =?$).

$$
\begin{aligned}
&IndJoin((S_1[w_1]), ..., (S_m[w_m]), C, o[, F]) = \\
&\quad \{(t, d_2, d_3, ..., d_n, Ind(S_1, t + w_1, o, d_2, d_3, ..., d_n)) \mid \\
&\qquad (t, d_2, d_3, ..., d_n, Ind(S_1, t + w_1, o, d_2, d_3, ..., d_n)) \in \\
&\qquad\quad IndSeq(S_1, C, o, w_1) \wedge \\
&\qquad Ind(S_1, t + w_1, o, d_2, d_3, ..., d_n) = F \wedge \\
&\qquad \forall\, (S, w) \in \{(S_2, w_2), ..., (S_m, w_m)\} : \\
&\qquad\quad (t, d_2, d_3, ..., d_n, Ind(S_1, t + w_1, o, d_2, d_3, ..., d_n)) \in \\
&\qquad\qquad IndSeq(S, C, o, w)\}
\end{aligned}
\tag{6.3}
$$

With these definitions, we can output all sentinels in the cube, *C*, with the offset, *o*, and the warning period, *w*, as shown in Formula 6.4. Each sentinel is represented as a set of source measures, *Source*, and a target measure, *Target*, and we use *MaxSource* as a threshold for the maximum number of source measures we want to combine in a sentinel. The thresholds β, γ, and σ are global and are thus not passed to Formula 6.4.

$$
\begin{aligned}
&SentRules(C, o, w) = \{Source_k \rightsquigarrow Target_l | \\
&\quad Source_k \subseteq \{M_1, ..., M_{p-1}, \overline{M_1}, ..., \overline{M_{p-1}}\} \wedge \\
&\quad |Source_k| \leqq MaxSource \ \wedge \ Target_l \in \{M_p, \overline{M_p}\} \wedge \\
&\quad SentSupp(Source_k, C, o) \geqq \sigma \wedge \\
&\quad ElimSupp(Source_k, Target_l, C, o, w, \blacktriangle) \geqq 0 \wedge \\
&\quad ElimSupp(Source_k, Target_l, C, o, w, \blacktriangledown) \geqq 0 \wedge \\
&\quad Balance(Source_k, Target_l, C, o, w) \geqq \beta \wedge \\
&\quad Confidence(Source_k, Target_l, C, o, w) \geqq \gamma\}
\end{aligned}
\tag{6.4}
$$

The functions *SentSupp* (Formula 6.5), *ElimSupp* (Formula 6.6), *Balance* (Formula 6.7), and *Confidence* (Formula 6.8) used in $SentRules(C, o, w)$ are defined as follows:

$$
SentSupp(Source, C, o) = |IndJoin(Source, C, o)| \tag{6.5}
$$

$$
\begin{aligned}
&ElimSupp(Source, Target, C, o, w, F) = \\
&\qquad |IndJoin(Source, (Target, w), C, o, F)| \\
&\qquad - |IndJoin(Source, (\overline{Target}, w), C, o, F)|
\end{aligned}
\tag{6.6}
$$

$$
\begin{aligned}
&Balance(Source, Target, C, o, w) = \frac{4 \times |A| \times |B|}{(|A| + |B|)^2} \\
&\text{where} \\
&\qquad A = ElimSupp(Source, Target, C, o, w, \blacktriangle) \\
&\qquad B = ElimSupp(Source, Target, C, o, w, \blacktriangledown)
\end{aligned}
\tag{6.7}
$$

$$
Confidence(Source, Target, C, o, w) = \frac{|A| + |B|}{SentSupp(Source, C, o)} \tag{6.8}
$$

Formulae 6.5 to 6.8 are based on the cardinality of indication sequences that result from joining indication sequences based on the measures in *Source* and *Target*.

In Table 6.2 we see the individual indication sequences for the measure, *PeoInv*, and the inverted measures $\overline{UniRev}$ and $\overline{Rev}$, from Table 6.1 (first three columns). In addition, we see the result of $IndJoin(PeoInv, \overline{UniRev}, (\overline{Rev}, 3), C, 1)$ that has a cardinality of 12, and $IndJoin(PeoInv, \overline{UniRev}, (\overline{Rev}, 3), C, 1, \blacktriangle)$ with the cardinality of 6. We note that $o = 1$ and $w = 3$ similar to our example in Section 6.2. In this particular case, the cardinalities represent *ElimSupp* and *A* respectively for the sentinel with $Source = \{PeoInv, \overline{UniRev}\}$ and $Target = \overline{Rev}$, since the cardinalities of the same joins, but for $Target = Rev$ are both 0.

$A = ElimSupp(Source, Target, C, o, w, \blacktriangle)$ (Formulae 6.7 and 6.8) is the number of times that *Source* changes in a certain direction and *Target* subsequently changes in the "expected" positive (▲) direction, minus the number of times where *Target* changes in the opposite direction. $ElimSupp(Source, Target, C, o, w, \blacktriangledown)$, denoted by *B*, is calculated in the same way, but for the changes to *Target* in the opposite direction of *A* (▼). We refer to this as the *contradiction elimination process*, where we essentially force a sentinel to be either $Source \rightsquigarrow Target$ or $Source \rightsquigarrow \overline{Target}$, and thereby we effectively eliminate both *contradicting* (same premise but different consequent) and *orthogonal* (different premise but same consequent) indications in the sentinel we are evaluating.

Formulae 6.5, 6.7, and 6.8 represent desired qualities of the sentinel: *SentSupp* (Formula 6.5) tells us how often the premise of the sentinel occurs. *Balance* (Formula 6.7) is used to determine the degree to which a generalized sentinel rule is uni-directional (*Balance=0*) or completely bi-directional (*Balance=1*), meaning that there are exactly the same amounts of positive and negative indications on the target measure in the data, similar to the sentinel, $PeoInv \wedge \overline{UniRev} \rightsquigarrow \overline{Rev}$, from our

IndSeq $(PeoInv, C, 1, 0)$	*IndSeq* $(\overline{UniRev}, C, 1, 0)$	*IndSeq* $(\overline{Rev}, C, 1, 3)$	*IndJoin* $(PeoInv, \overline{UniRev}, (\overline{Rev}, 3), C, 1)$	*IndJoin* $(PeoInv, \overline{UniRev}, (\overline{Rev}, 3), C, 1, \blacktriangle)$
2009-Jan,▲		2009-Jan,▼		
	2009-Feb,▲	2009-Feb,▼		
2009-Mar,▲	2009-Mar,▲	2009-Mar,▲	2009-Mar,▲	2009-Mar,▲
2009-Apr,▼	2009-Apr,▼	2009-Apr,▼	2009-Apr,▼	
2009-May,▲	2009-May,▲			
2009-Jun,▼	2009-Jun,▼	2009-Jun,▼	2009-Jun,▼	
2009-Jul,▲	2009-Jul,▲	2009-Jul,▲	2009-Jul,▲	2009-Jul,▲
2009-Aug,▲	2009-Aug,▲	2009-Aug,▲	2009-Aug,▲	2009-Aug,▲
2009-Sep,▲	2009-Sep,▲	2009-Sep,▲	2009-Sep,▲	2009-Sep,▲
2009-Oct,▼	2009-Oct,▼	2009-Oct,▼	2009-Oct,▼	
2009-Nov,▲	2009-Nov,▲	2009-Nov,▲	2009-Nov,▲	2009-Nov,▲
2009-Dec,▲	2009-Dec,▲	2009-Dec,▲	2009-Dec,▲	2009-Dec,▲
2010-Jan,▼	2010-Jan,▼	2010-Jan,▼	2010-Jan,▼	
2010-Feb,▼	2010-Feb,▼	2010-Feb,▼	2010-Feb,▼	
2010-Mar,▼	2010-Mar,▼	2010-Mar,▼	2010-Mar,▼	

Table 6.2: Example of Indication Sequences and Joins.

example in Section 6.2. *Confidence* (Formula 6.8) tells us the fraction of occurrences where the premise occurs, and the consequent occurs within w time. We denote the minimum threshold for *SentSupp* by σ, the minimum threshold for *Confidence* is denoted by γ, and the minimum threshold for *Balance* is denoted by β.

Aside from the individual quality measures for a sentinel in Formulae 6.5 to 6.8, it is also desirable to have a quality measure that incorporates all these measures into one value; this is relevant if we want to compare multiple different sentinels to identify the best sentinel(s). For this purpose, we define *Score* for a sentinel, $Source \rightsquigarrow Target \in SentRules(C, o, w)$, as shown in Formula 6.9.

$$\begin{aligned} &Score(Source, Target, C, o, w) = \\ &\quad (1 - wp + \frac{(1 + Maxw - w) \times wp}{Maxw}) \\ &\quad \times (\frac{1}{2} + \frac{1 + MaxSource - |Source|}{MaxSource \times 2}) \\ &\quad \times \frac{ElimSupp(Source, Target, C, o, w, ?)}{MaxElimSupp(C, o, w)} \\ &\quad \times Confidence(Source, Target, C, o, w) \\ &\quad \times (\frac{1}{2} + \frac{Balance(Source, Target, C, o, w)}{2}) \end{aligned} \tag{6.9}$$

With this definition of *Score*, we denote the maximal value of $ElimSupp(Source, Target, C, o, w, ?)$ for any sentinel, $Source \rightsquigarrow Target \in SentRules(C, o, w)$, by $MaxElimSupp(C, o, w)$. In addition, we introduce the threshold, *Maxw*, which is the maximum length of the warning period, w, we are willing to accept. The constant, *wp*, represents the warning penalty, i.e., the degree to which we want to penalize rules with a higher *w* (0=no penalty, 1=full penalty). Generally, *Score* incorporates a preference having high values for all quality measures (Formulae 6.5 to 6.8), and having shorter sentinel rules (low $|Source|$) with a low, non-cyclic warning period, w. The construction of Score is further elaborated in Chapters 3 and 5. With *Score* as a uniform way to assess the quality of a sentinel, we can now define *Optimalw*(C,o), as shown in Formula 6.10, which is the value of *w*, $1 \leqq w \leqq Maxw$, where *SentRules*(C,o,w) contains the rule with the highest *Score*.

$$\begin{aligned} &Optimalw(C, o) = w \text{ such that } 1 \leqq w, w' \leqq Maxw \ \wedge \\ &\quad \exists S \in SentRules(C, o, w) : \\ &\quad\quad (\forall w' \neq w : (\forall S' \in SentRules(C, o, w') : \\ &\quad\quad\quad (Score(S) \geqq Score(S')))) \end{aligned} \tag{6.10}$$

$$\begin{aligned} &SentRulesPruned(C, o, w) = \{S \in SentRules(C, o, w) \mid \\ &\nexists S' \in SentRules(C, o, w) : \\ &(Score(S', C, o, w) \geqq Score(S, C, o, w) \ \wedge \ S'_{Source} \subset S_{Source})\} \end{aligned} \tag{6.11}$$

Having found the optimal *w*, it is also desirable to prune the generalized sentinel rules such that we only output the best rules in terms of *Score*, and the shortest rules in

SentRules	*RuleLen*	*SentSupp* (*ElimSupp*)	*Conf*	*Balance*	*Score*	*OK* ?
$PeoInv \wedge \overline{UniRev} \rightsquigarrow \overline{Rev}$	2	13 (12)	92%	1	0.71	OK
$PeoInv \rightsquigarrow \overline{Rev}$	1	14 (11)	79%	0.99	0.66	OK
$UniRev \rightsquigarrow Rev$	1	14 (11)	79%	0.99	0.66	OK
$PeoInv \wedge WHts \rightsquigarrow \overline{Rev}$	2	7 (6)	86%	1	0.33	OK
$PeoInv \wedge \overline{UniRev} \wedge WHts \rightsquigarrow \overline{Rev}$	3	7 (6)	86%	1	0.26	OK
$UniRev \wedge \overline{WHts} \rightsquigarrow Rev$	2	8 (5)	63%	0.96	0.19	Failed
$WHts \rightsquigarrow \overline{Rev}$	1	11 (2)	18%	1	0.03	Failed

Table 6.3: Sentinels ordered by their respective *Conformance* and *Score*.

terms of number of source measures. For this purpose, we use the *SentRulesPruned* function, as shown in Formula 6.11, that eliminates rules with poor quality (lower *Score*) if a shorter rule exists with at least as good a *Score*, and where *Source* is a proper subset of *Source* for the longer rule.

We say that $SentRulesPruned(C, o, Optimalw(C, o))$ ordered by their respective *Score* are the best sentinels in a database, *C*, with the offset, o. Using the *SentRulesPruned* function, we note that $PeoInv \wedge WHts \rightsquigarrow \overline{Rev}$ and $PeoInv \wedge \overline{UniRev} \wedge WHts \rightsquigarrow \overline{Rev}$ in fourth and fifth line in Table 6.3 would be eliminated since the shorter rules $PeoInv \rightsquigarrow \overline{Rev}$ and $PeoInv \wedge \overline{UniRev} \rightsquigarrow \overline{Rev}$ have a better *Score*. In other words, we do not improve the quality of these sentinels by adding the source measure *WHts* to them.

We use the following notation when describing a sentinel, $Source \rightsquigarrow Target \in SentRules(C, o, w)$: we write all source measures in *Source* and *Target* as one string where the source measures are separated with $\wedge$ and the target measure separated with $\rightsquigarrow$, e.g., $A \wedge B \rightsquigarrow C$. The total number of potential sentinels for the p measures in the dataset, *C*, with *MaxSource* source measures is $\sum_{x=1}^{l} \frac{(2p)!}{(2p-x)!}$ where $l = MaxSource$, however by preserving the order of the source measures, and never allowing the first source measure to be inverted, we can reduce the number of potential rules (permutations) to $\sum_{x=1}^{l} \frac{(2p-1)!}{l!(2p-1-x)!}$ without loosing any sentinels from a logical perspective. Therefore, we preserve the order of the source measures and invert measures relative to the first source measure when describing sentinels.

If we apply Formula 6.4 to the data in Table 6.1, and use the notation above, we get the conforming sentinels (Conformance=OK) in the first column of Table 6.3 as output when the thresholds for number of source measures, *SentSupp*, *Balance*, and *Confidence* are set as follows: $\alpha = 10\%$, $\sigma = 5$, $\beta = 0.8$, and $\gamma = 75\%$, $MaxSource = 3$. Furthermore, $wp = \frac{1}{2}$.

6.4 Prerequisites for Bitmapped Sentinel Mining

To mine the sentinels in a dataset, we need to test all combinations of source measures up to *MaxSource* in combination with the target measure and warning periods allowed. The measure combinations include inverted measures as well. For each sentinel, based on these combinations, we need to inspect whether it conforms with the thresholds set for the desired sentinel qualities. To do this, we need to join (Formula 6.3) a number of indication sequences (Formula 6.2) and subsequently do a number of calculations based on the cardinality of the resulting indication sequences (Formulae 6.4 to 6.11). Bitmapped sentinel mining optimizes this process by encoding all indication sequences into bitmaps, this means that indication sequence joins can be done in a number of Boolean operations. Subsequently, the cardinality of the resulting bitmaps can be found by counting the bits set in these bitmaps.

The *SentBit* algorithm first encodes the data into bitmaps, and from this point a recursive function is initiated that *grows* the potential sentinels in terms of number of source measures with respect to the order of source measures and the thresholds set. Upon completion of the recursive function, the best sentinels can be output from memory. Overall the *SentBit* algorithm can be described as follows:

Step 1: Use *EncodeData* and *CreateBitmap* to build a bitmap for each source measure, and for the target measure for each warning period allowed.
Step 2: Call *TestSentinel* for each combination of source measure, and target measure with a given warning period allowed. *TestSentinel* will call itself recursively by adding source measures until *Score* does not improve, or the number of source measures meet the maximum length desired. Whenever a sentinel is tested in *TestSentinel*, it is stored in memory if it meets the quality thresholds.
Step 3: Output the sentinels for the warning period that includes the highest scoring sentinel.

A bitmap used for bitmapped sentinel mining is a list of an even number of CPU words. The *SentBit* algorithm presented next requires that the indication sequences for all measures are first encoded into bitmaps of equal length in terms of number of words. This encoding will allows us to conduct the entire sentinel mining process as a number of Boolean operations on the bitmaps with subsequent counts of the number of bits set. To illustrate the encoding and bitmap operations, we use the indication sequence for the source measure *PeoInv*, formally $IndSeq(PeoInv, C, 1, 0)$, as shown in Table 6.2.

Bitmap($PeoInv$**)** aligned in two 32 bit words

bits set for positive *Ind* | fill of 0 bits to align with word length

```
PI : 10101011 1011000  0 00000000 00000000
NI : 00010100 0100111  0 00000000 00000000
```

bits set for negative *Ind* | fill of 0 bits to align with word length

As shown in *Bitmap(PeoInv)*, the bits are set in accordance with the positive indications, *PI*, and the negative indications, *NI*. The position of the indications are given by the order of the tuple, $(t, d_2, d_3, ..., d_n) \in C$, thus even if there is no indication (positive or negative), $Ind(PeoInv, t, o, d_2, d_3, ..., d_n)$, for a given $(t, d_2, d_3, ..., d_n)$, the position will still be occupied by two unset bits for PI and NI, respectively. In our example, there is only a total of 15 possible indications which easily fits into the 32 bit word. In practice, however, PI and NI will most likely need multiple words allocated to fit the number of bits equivalent to the number of $(t, d_2, d_3, ..., d_n) \in C$. Similarly, we have the bitmaps for the indication sequences for the source measure, ***UniRev***, and the target measure, ***Rev***, formally $IndSeq(Unirev, C, 1, 0)$ and $IndSeq(Rev, C, 1, 3)$:

Bitmap($UniRev$**)** aligned in two 32 bit words

```
PI : 00010100 01001110 00000000 00000000
NI : 01101011 10110000 00000000 00000000
```

Specifically for the target measure, we create a bitmap for each warning period we want to evaluate, i.e. $1, 2, 3, ..., Maxw$, thus we denote the bitmap for ***Rev*** with a warning period of 3 months as Bitmap(Rev_3).

Bitmap(Rev_3**)** aligned in two 32 bit words

```
PI : 11010100 01001110 00000000 00000000
NI : 00100011 10110000 00000000 00000000
```

The encoding of all measures into bitmaps is done using the *EncodeData* (Algorithm 6.1) procedure with its subroutine *CreateBitmap* (Algorithm 6.2).

Algorithm 6.1 Encode all measures into bitmaps

EncodeData(a set of source measures, $\{S_1, ..., S_p\} \subseteq C$, a target measure, $Target \in C$, an offset, o, and a maximum warning period, *Maxw*)

1: **for all** $S \in \{S_1, ..., S_p\}$ **do**
2: $\quad CreateBitmap(S, o, 0)$
3: **for** $w = 1$ to $Maxw$ **do**
4: $\quad CreateBitmap(Target_w, o, w)$

Algorithm 6.2 Create a bitmap for a measure

CreateBitmap(a measure, $M \in C$, an offset, o,
a warning period, w)
1: Allocate memory for $Bitmap(M)$, fill(0)
2: $position \leftarrow 0$
3: **for all** $(t + w, d_2, d_3, ..., d_n,$
$t + w + o, d_2, d_3, ..., d_n) \in C$ **do**
4: **if** $Ind(M, t + w, o, d_2, d_3, ..., d_n) =$ ▲ **then**
5: set bit # ($position$ **MOD** word length)
in $Bitmap(M).PI.word(position$ **DIV** word length)
6: **else if** $Ind(M, t + w, o, d_2, d_3, ..., d_n) =$ ▼ **then**
7: set bit # ($position$ **MOD** word length)
in $Bitmap(M).NI.word(position$ **DIV** word length)
8: $position \leftarrow position + 1$

EncodeData simply takes each source measure and each w combined with the target measure and calls *CreateBitmap* to do the encoding. Once called, *CreateBitmap* inspects all possible indications (Formula 6.1) for the measure, M, in the dataset, C, with the offset, o for the given warning period, w. We note that for a source measure $w = 0$, and for a target measure $1 \leqq w \leqq Maxw$. After encoding, the source measures will all be aligned such that all indications that occur for a given $(t, d_2, d_3, ..., d_n)$ are described by bits with the *same* position in either PI or NI of the bitmaps. At the same time, these indications are have the same position as the bits representing the target measure indications for a period, $w \mid 1 \leqq w \leqq Maxw$, later. This means that comparing the indications of the source measures with the target measure indications at a given value of w can be done simply by comparing a single bitmap for each measure.

For simplicity the following variables are, if not passed as parameters, global constants and are thus accessible for all sub-algorithms of the *SentBit* algorithm: the dataset, C, the source measures, $\{S_1, ..., S_p\} \subseteq C$, the offset, o, and the thresholds, α, β, γ, σ, *Maxw*, and *MaxSource*. We note that an inverted measure does not require encoding a new bitmap since the "inverted" bitmap is similar to the original measure in terms of the position of set bits in PI and NI, but the parts have been switched, e.g, $Bitmap(PeoInv).PI = Bitmap(\overline{PeoInv}).NI$ and vice versa. Therefore, we can do all the comparing of indications for an inverted measure simply by swapping PI with NI.

6.5 The SentBit Algorithm

Using the prerequisite encoding of measures into bitmaps, a bitmap is equivalent to an indication sequence (Formula 6.2), thus a join of two indication sequences (Formula 6.3) can be done using an AND operation on their bitmaps since the positions are aligned, as shown in the function *BitAND* (Algorithm 6.3). Using *BitAND* pairwise on the bitmaps for $PeoInv$, $\overline{UniRev}$, and $\overline{Rev_3}$ ($\overline{Rev}$ for $w = 3$), we can create a bitmap for ***IndJoin***$(PeoInv, \overline{UniRev}, (\overline{Rev}, 3), C, 1)$ in Table 6.2 as follows:

Bitmap($PeoInv$**)**
`PI : 10101011 10110000 00000000 00000000`
`NI : 00010100 01001110 00000000 00000000`
AND
Bitmap($\overline{UniRev}$**)**
`PI : 01101011 10110000 00000000 00000000` } swapped AND
`NI : 00010100 01001110 00000000 00000000` }
Bitmap($\overline{Rev_3}$**)**
`PI : 00100011 10110000 00000000 00000000` } swapped
`NI : 11010100 01001110 00000000 00000000` }
=
Bitmap($IndJoin(PeoInv, \overline{UniRev}, (\overline{Rev}, 3), C, 1)$**)**
`PI : 00100011 10110000 00000000 00000000`
`NI : 00010100 01001110 00000000 00000000`

Algorithm 6.3 AND of two bitmaps

BitAND(bitmap *A*, bitmap *B*)
1: Allocate size(*A*) memory for ***BitAND*** {size in words}
2: **for all** $word_i \in A$ **do**
3: $BitAND.word_i \leftarrow A.word_i$ AND $B.word_i$
4: **return** bitmap ***BitAND***

Algorithm 6.4 Count the bits set in a bitmap

BitCount(bitmap ***Bits***, $Parts \subseteq \{PI, NI\}$)
1: $BitCount \leftarrow 0$
2: **for all** $P \in Parts$ **do**
3: **for all** $word_i \in Bits.P$ **do**
4: $BitCount \leftarrow BitCount +$ PopCount$(word_i)$
5: **return** ***BitCount***

The final ingredient needed to assess a sentinel is the ability to count the number of set bits in a bitmap, which is equal to the cardinality of a given indication sequence or join. This is done using the *BitCount* function (Algorithm 6.4). In *BitCount* we assume the availability of a *Population Count (PopCount)* function that counts the number of bits set in a word, in Section 6.6 we describe three different approaches to this task. As seen in *BitCount* it is an uncomplicated task to summarize the *PopCount* of all words in a bitmap. *BitCount* also allows for a parameter, *Parts*, to be passed which allows us to count the bits set in PI, NI or both.

Using the sub-algorithms, we are now able to run the sentinel mining process using bitmaps, *SentBit* (Algorithm 6.5), that calls the recursive function *TestSentinel* (Algorithm 6.6). The following arrays are globally available to all sub-algorithms in order to store the mining results as the recursion progresses: *SentList* is a dynamic array of (a set of measures *Source*, measure *Target*, *NewScore*, *w*); *MaxElimSupp* is an array of *Maxw* values (Default 0).

Algorithm 6.5 *SentBit*: Sentinel Mining Using Bitmaps

Input: the max number of sentinels to be returned, n,
a dataset, C, a set of source measures, $\{S_1, ..., S_p\} \subseteq C$, a target measure, $Target \in C$, an offset, o, and the thresholds, α, β, γ, σ, *Maxw*, and *MaxSource*

1: *EncodeData*($\{S_1, ..., S_p\}$, $Target, o, Maxw$)
2: Allocate memory for *SentList*
3: Allocate memory for *MaxElimSupp*
4: **for** $w = 1$ to $Maxw$ **do**
5: **for** $x = 1$ to p **do**
6: *TestSentinel*($x, \emptyset, Bitmap(w)_{AllBitsSet}, Target, w, 0$)
7: **return** the top n sentinels from *SentList* for the value of w where the sentinel with the highest *NewScore* exists. In the output $Score \leftarrow \frac{NewScore}{MaxElimSupp(w)}$

Upon receipt of the desired number of sentinels, n, the dataset, C, the source and target measures, the offset, o, and the thresholds, the algorithm initiates the mining process by encoding the relevant measures into bitmaps in line 1. *SentBit* allocates memory for the output of the mining process (lines 2 and 3), and from here *TestSentinel* (Algorithm 6.6) is initiated by intuitively adding a) a source measure to a sentinel with b) the target measure, c) no previous source measures, d) a bitmap with all bits set (except for the bits at the endpoint periods that are not to be evaluated for a given w), e) the specific warning period, and f) a $Score = 0$, for each $1 \leqq w \leqq Maxw$ (lines 4 to 6). Upon completion of the recursive mining process, the process of outputting maximum n sentinels found (Formula 6.11), for the value of w where the largest *Score* exists (Formula 6.10), and sorted descending by *Score* is trivial (line 7). We note that the outstanding division of *NewScore* by $MaxElimSupp(w)$

in accordance with Formula 6.9 is done only on the maximum n sentinels output (see explanation of the *CalcScore* function, Algorithm 6.7).

Algorithm 6.6 Test for adding a source measure to a sentinel

TestSentinel(source measure number *AddSource*,
set of measures *Source*, bitmap *Bits*,
a measure *Target*, *w*, *Score*)

1: **for all** $S \in \{S_{AddSource}, \overline{S_{AddSource}}\}$ **do**
2: $NewBits \leftarrow BitAND(Bitmap(S), Bits)$
3: **if** *BitCount(NewBits,*$\{$*PI,NI*$\}$*)<BitCount(Bits,*$\{$*PI,NI*$\}$*)* **then**
4: $NewSource \leftarrow Source \cup S$
5: **for all** $T \in \{Target, \overline{Target}\}$ **do**
6: $CalcScore(NewBits,$
$BitAND(Bitmap(T_w), NewBits),$
$BitAND(Bitmap(\overline{T_w}), NewBits),$
$|NewSource|, w)$
7: **if** *NewScore>Score* **then**
8: **if** $SentSupp \geqq \sigma \ \wedge \ Confidence \geqq \gamma \wedge$
$Balance \geqq \beta$ **then**
9: Append $(NewSource \rightsquigarrow T, NewScore, w)$ to *SentList*
10: **if** $|NewSource| < MaxSource$ **then**
11: **for** $y = AddSource + 1$ to p **do**
12: $TestSentinel(y, NewSource, NewBits,$
$T, w, NewScore)$
13: **if** $(A + B) > MaxElimSupp(w)$ **then**
14: $MaxElimSupp(w) \leftarrow (A + B)$

The *TestSentinel* function (Algorithm 6.6) tests if adding the source measure, $S_{AddSource}$, to a given set of source measures, *Source*, would improve the sentinel relationship with the target measure, *Target*. Since *TestSentinel* is recursive, we potentially need the bitmap representing the source measures many times during the recursion, and to optimize this we apply a standard dynamic programming approach by passing the bitmap, *Bits*, representing the AND operation of all bitmaps for the measures in *Source*. In addition, the function receives the *Score* for the sentinel represented by *Source* and *Target* without $S_{AddSource}$. Finally, the warning period for the sentinel, w is also passed along.

In *TestSentinel*, there are two options for adding the additional source measures, namely directly, $S_{AddSource}$, or inverted, $\overline{S_{AddSource}}$, as seen in line 1. The bitmap for the added source measure is joined (Algorithm 6.3) with the bitmap for the existing source measures, *Bits*, in line 2, and the result is evaluated to have less bits set than

before in line 3. If the *BitAND* does not result in *fewer* bits, then the added source measure is exactly equal to the source measures that are already in the sentinel, meaning that an equally good sentinel can be created with the new source measure alone, and thus it is irrelevant to continue with the longer sentinel at hand. We note that sending a bitmap, *Bits*, will all bits set (both PI and NI) will satisfy the condition in line 3 for the any source measure during the initial recursion call from *SentBit* (Algorithm 6.5, line 6). If the added source measure did indeed reduce the number of bits, there are again two options to create a conforming (to thresholds) sentinel: either with $Target_w$ or $\overline{Target_w}$. Although there is a good chance that a sentinel will have the same direction on the target measure even though it is made more specific by adding a source measure, we cannot know this for sure, e.g., if adding the source measure reduces the number of indications covered the sentinel by more than 50%. With the two combinations of target measures, we use the function *CalcScore* (Algorithm 6.7) to calculate and test if the *Score* for the new potential sentinel, *NewScore*, is better than *Score* from its ancestor (lines 6 and 7). If this is the case, we record the sentinel in *SentList* if it meets the defined thresholds (lines 8 and 9). In addition, if the sentinel has less than *MaxSource* source measures it is a candidate for growing another and better sentinel by combining it with one of the remaining source measures and *TestSentinel* will call itself to grow the sentinel further (lines 10 to 12). We note that meeting the thresholds is not a criteria for growing the sentinel further; as long as the *Score* is improving we have the potential for growing a sentinel that will eventually meet the thresholds. Also, we note that we will never add a source measure that occurred prior to $S_{AddSource}$ in $\{S_1, ..., S_p\} \subseteq C$, this means that we are preserving the order of the source measures, and we are thus reducing the number of tested combinations as explained earlier in Section 6.3. Finally, when a potential sentinel is found to improve the Score, we also test if it is a candidate to represent the value for $MaxElimSupp(C, o, w)$ (lines 13 and 14), and if $A + B$ is better than the last candidate for $MaxElimSupp(C, o, w)$, stored in the array $MaxElimSupp(w)$, it is replaced with $A + B$.

The function *CalcScore* calculates the *Score* for a potential sentinel (excluding division by the constant $MaxElimSupp(C, o, w)$ as explained above). A sentinel can be represented by a bitmap representing the AND operation of all source measure bitmaps, *SourceBits*, and two bitmaps representing an AND operation of *SourceBits* and the bitmaps for a target measure, *TargetBits*, and the opposite of this target measure, *ElimBits*, respectively. The term "opposite" is used deliberately since the target measure can also be inverse, thus the opposite in this case would be the original target measure.

As seen in lines 1 to 3, *CalcScore* (Algorithm 6.7) calls *BitCount* (Algorithm 6.4) to identify *SentSupp*, A, and B, and once identified, calculating *Confidence* (line 4) and *Balance* (line 5) is trivial. *CalcScore* also gets the parameter, *SourceCnt*, which

Algorithm 6.7 Calculate quality measures

CalcScore(bitmap *SourceBits*, bitmap *TargetBits*,
bitmap *ElimBits*, *SourceCnt*, w)

1: $SentSupp \leftarrow BitCount(SourceBits, \{PI, NI\})$
2: $A \leftarrow BitCount(TargetBits, \{PI\})$
$- BitCount(ElimBits, \{PI\})$
3: $B \leftarrow BitCount(TargetBits, \{NI\})$
$- BitCount(ElimBits, \{NI\})$
4: $Confidence \leftarrow \frac{|A|+|B|}{SentSupp}$
5: $Balance \leftarrow \frac{4\times|A|\times|B|}{(|A|+|B|)^2}$
6: $NewScore \leftarrow (1 - wp + \frac{(1+Maxw-w)\times wp}{Maxw})$
$\times (\frac{1}{2} + \frac{1+MaxSource-SourceCnt}{MaxSource\times 2})$
$\times (A + B) \times Confidence$
$\times (\frac{1}{2} + \frac{Balance}{2})$
7: **return** *NewScore*, *SentSupp*, *Confidence*, *Balance*, A, B

is the number of source measures that constitutes *SourceBits*, and in addition it gets w as a parameter. These values are used to calculate *NewScore* (line 6) which is equal to Score (Formula 6.9) with the exception of the division by $MaxElimSupp(C, o, w)$, since it is constant and thus irrelevant to the ranking of sentinels. Moreover, $MaxElimSupp(C, o, w)$ is not known until we have inspected all sentinels in C for a given value of o and w. Therefore, the scores output for the sentinels mined are adjusted to comply with Formula 6.9 upon finalization of the *SentBit* algorithm (Algorithm 6.5, line 7).

To exemplify the operations in *CalcScore*, we refer to the sentinel $PeoInv \wedge \overline{UniRev} \rightsquigarrow \overline{Rev}$ in Table 6.3 with a warning period of 3 months that originated from the data in Table 6.1. The prerequisite bitmaps of *CalcScore* are: a bitmap where the source measure indication sequences are joined, *SourceBits*, a bitmap where the source measure indication sequences and the target measure indication sequence are joined, *TargetBits*, and a bitmap where the source measure indication sequences and the target measure indication sequence in the opposite direction are joined, *ElimBits*. *SourceBits* are given as $IndJoin(PeoInv, \overline{UniRev}, C, 1)$:

Bitmap($PeoInv$**)**
PI : 10101011 10110000 00000000 00000000
NI : 00010100 01001110 00000000 00000000
AND
Bitmap($\overline{UniRev}$**)**
PI : 01101011 10110000 00000000 00000000
NI : 00010100 01001110 00000000 00000000
= *SourceBits*
PI : 00101011 10110000 00000000 00000000 (7 bits)
NI : 00010100 01001110 00000000 00000000 (6 bits)

$TargetBits = IndJoin(SourceBits, (\overline{Rev}, 3), C, 1) =$
$IndJoin(PeoInv, \overline{UniRev}, (\overline{Rev}, 3), C, 1)$ as exemplified in Section 6.5, thus:

TargetBits
PI : 00100011 10110000 00000000 00000000 (6 bits)
NI : 00010100 01001110 00000000 00000000 (6 bits)

Finally, $ElimBits = IndJoin(SourceBits, (Rev, 3), C, 1)$ since the opposite of $\overline{Rev}$ is *Rev*:

SourceBits
PI : 00101011 10110000 00000000 00000000
NI : 00010100 01001110 00000000 00000000
AND
Bitmap(Rev_3**)**
PI : 11010100 01001110 00000000 00000000
NI : 00100011 10110000 00000000 00000000
= *ElimBits*
PI : 00000000 00000000 00000000 00000000 (0 bits)
NI : 00000000 00000000 00000000 00000000 (0 bits)

Using *BitCount* on *SourceBits*, *TargetBits*, and *ElimBits* as shown in *CalcScore* (Algorithm 6.7, lines 1-3), we find that $SentSupp = 7+6 = 13$, $A = 6-0 = 6$, and $B = 6-0 = 6$ for the sentinel $PeoInv \wedge \overline{UniRev} \rightsquigarrow \overline{Rev}$, which is of course similar to our example in Section 6.2 on the Table 6.1 data. From this point, completing the remaining calculations in *CalcScore* is trivial (lines 4-6).

This concludes the walk-through of the *SentBit* algorithm. In the following section we will elaborate the different optimization options for the *SentBit* algorithm.

In Section 6.7 we will demonstrate that *SentBit* is superior in performance compared to prior art (Chapters 3 and 5), and in addition we will show the effect of different optimization strategies.

6.6 Optimization and Implementation

In addition to the inherent optimization over prior art (Chapter 3) with bitmaps, we use three different strategies to optimize the *SentBit* algorithm even further, namely: *pruning*, *dedicated processor instructions*, and *parallelization*.

Pruning: Similar to the RPG functionality in Chapter 3 we can apply an Apriori-style optimization by pruning the measures that will never be able to be part of a conforming sentinel. Specifically, we know that in order for a source measure, or a target measure for a given w, to be part of a conforming sentinel, it needs to have at least σ indications in its indication sequence (to fulfill $SentSupp \geqq \sigma$). Therefore, we can simply dismiss any bitmap, $Bitmap(M)$, in *CreateBitmap* where $BitCount(Bitmap(M), \{PI, NI\}) < \sigma$. This can be implemented simply by appending one line to *CreateBitmap* (Algorithm 6.2) as follows:

if $BitCount(Bitmap(M), \{PI, NI\}) < \sigma$ **then**
  Free $Bitmap(M)$

In addition, line 6 in *SentBit* (Algorithm 6.5) and line 12 in *TestSentinel* (Algorithm 6.6) needs to verify whether the bitmap is allocated or freed, and in the latter case skip the measure. In Section 6.7 we demonstrate the effect of pruning, and we denote the algorithm using this optimization by "Prune".

Dedicated processor instructions: As mentioned in conjunction with the *BitCount* function in Section 6.5, there are different options for counting the number of bits set in a CPU word (Algorithm 6.4, line 4). The instruction known as *population count* (POPCNT) returns the number of bits set, and it is part of the SSE4a instruction set available, e.g., on the Intel "Core i7" [29] and the AMD "10h" [1] processor families. We denote the algorithm using this approach by "PopCnt". As an alternative to POPCNT, we can create an array where each element holds the precalculated bit count, e.g., an array, A, of 65536 bytes $(0...65535)$ where the values $A(0) = 0$ and $A(65535) = 16$. In this case, the bit count of each 16 bits in the CPU word can be found through a single lookup. The size of the array should be limited to fit the CPU cache, and thus with contemporary processors this approach is not feasible for

lookups of more than 16 bits at the time. We denote the algorithm using this approach by "Array". Whenever we do not use "PopCnt" or "Array", we apply a loop equal to the length of a CPU word that counts the number of times LSB is set when shifting the bits right (SHR). We denote the algorithm using this approach by "Basic" when running without any other optimization. It should be noted that this approach is also used in the "Prune" and "Parallel" variants. The effect of the "PopCnt" and "Array" optimization strategies compared to "Basic" is demonstrated in Section 6.7.

Parallelization: We apply a parallelization strategy where *SentBit* (Algorithm 6.5, line 6) is parallelized by sending each initial recursion for a source measure, target measure, and warning period combination (*TestSentinel*$(x, \emptyset, Bitmap(w)_{AllBitsSet}, Target, w, 0)$) to an available core. We denote the algorithm using parallelization by "Parallel" in Section 6.7.

Best-of-Breed: the "best-of-breed" variant, denoted by "Best", incorporates the best optimization choice with regards to pruning, dedicated processor instructions, and parallelization based on our experiments (Section 6.7). Specifically, the "Best" variant combines the "Prune", "PopCnt", and "Parallel" optimization options.

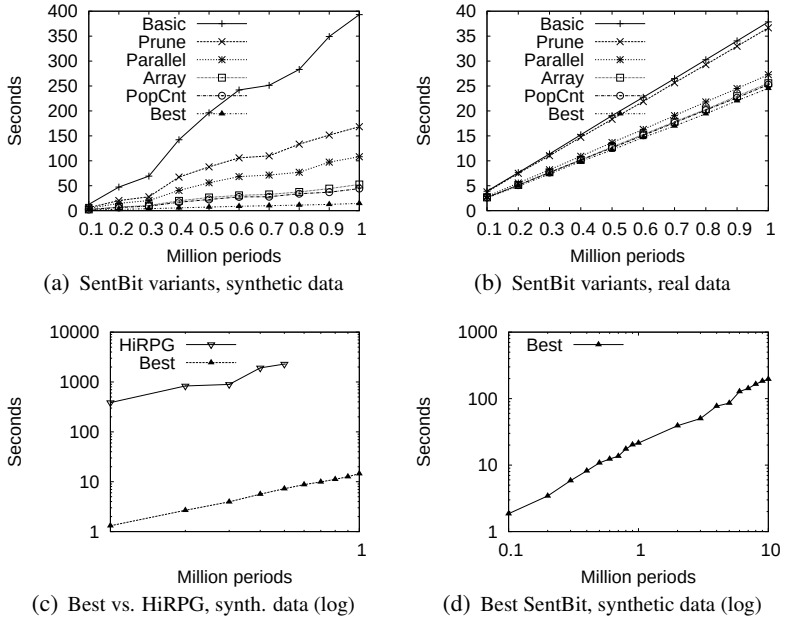
Implementation: The algorithm variants were implemented in C++ and compiled into a stand-alone 64-bit executable file using Microsoft Visual Studio 2010. The different optimizations were in-lined in the code such that no instructions were used to select optimization method during the execution of the algorithm. In order to access the processor specific POPCNT instruction, we used a so-called *intrinsic* function.

Computational Complexity: When we examine the *SentBit* algorithm, we note that the computational complexity will be dominated by the encoding of bitmaps, Algorithm 6.1, and the recursive mining process initiated in Algorithm 6.5. The encoding of bitmaps can be performed in time $\mathcal{O}(n \times (p + m))$, where *n* is the number of periods, *p* is the number of source measures, and *m* is *Maxw*. The recursive mining process can be performed in time $\mathcal{O}(\frac{n}{WL} \times p^l \times m)$, where *WL* is the CPU word length in bits and *l* is *MaxSource*. Therefore *SentBit* has a complexity of $\mathcal{O}(n + (\frac{n}{WL} \times p^l \times m))$, since $p^l \times m >> p + m$. Depending on whether $n >> p^l \times m$ or not, the computational complexity will be dominated by either the encoding or the recursive mining process. In Section 6.7 we test this assessment through extensive experiments.

6.7 Experiments

Setup: We run our experiments on an Intel Core i7 Quad CPU (920) 2.66GHz PC with 12GB RAM and 2 500GB disks (7,200 RPM) running a 64Bit version of Windows 2008 Server R2. We use a range of synthetic datasets and a range of real-world datasets for the experiments. The synthetic datasets have 150 source measures and were created using three simple random walks such that 30 source measures have 80% chance of changing α, up or down, 70 source measures have 40% chance of changing α, up or down, and the remaining 50 source measures do not change above α. In our experience with real-world data, having a dataset where $\frac{1}{5}$ of the measures change many times, whereas $\frac{1}{3}$ of the measures have no relevant changes, is very realistic. The target measure was aligned such that it always changed one period later synchronized with the first source measure with many changes. This means that there will always exist at least one sentinel with $w = 1$ in these data. The synthetic datasets range from 100,000 to 1,000,000 rows in 100,000 row intervals, and from 1,000,000 to 10,000,000 rows in 1,000,000 row intervals. We note that the sizes of these datasets are *huge* compared to the real-world dataset. In general, we would expect any real application of sentinels to work on *significantly fewer rows* since we typically aggregate the data, e.g., into months or weeks, before mining sentinels. The real-world datasets are produced from the operational data warehouse of TARGIT A/S. Based on experience with more than 3,800 customers worldwide, we will characterize this dataset as typical for a medium-sized company with a mature data warehouse. The original dataset contains 241 months (20.1 years) of operational data scattered across 148 source measures. We created descendants of the real-world dataset from 100,015 to 999,909 rows in $\approx 100,000$ row intervals by duplication. Descendants of the synthetic datasets are produced by selecting the source measure "tied" to the target measure, and the remaining number of source measures randomly to produce datasets with 10, 20, 30, ..., 150 source measures. Descendants of the real-world datasets are produced similarly, but with all source measures selected randomly. When nothing else is specified, we either use the original real-world dataset with all 148 source measures, or the synthetic dataset with 100,000 rows and 30 randomly selected source measures. We use the following algorithm settings: *wp*=0.5, *MaxSource*=3, *Pareto*=85% (applies to HiRPG only), and thresholds: *SentSupp*=5, *Conf*=0.8, *Balance*=0.8, and *Maxw*=12.

Scaling dataset size: In Figure 6.1 we validate that all variants of *SentBit* scales linearly to 1,000,000 periods of synthetic and real data (when the other factors are kept constant). In addition, we validate that the "Best" variant scales linearly to 10,000,000 periods of synthetic data. The slight synchronous variance in linearity



(a) SentBit variants, synthetic data

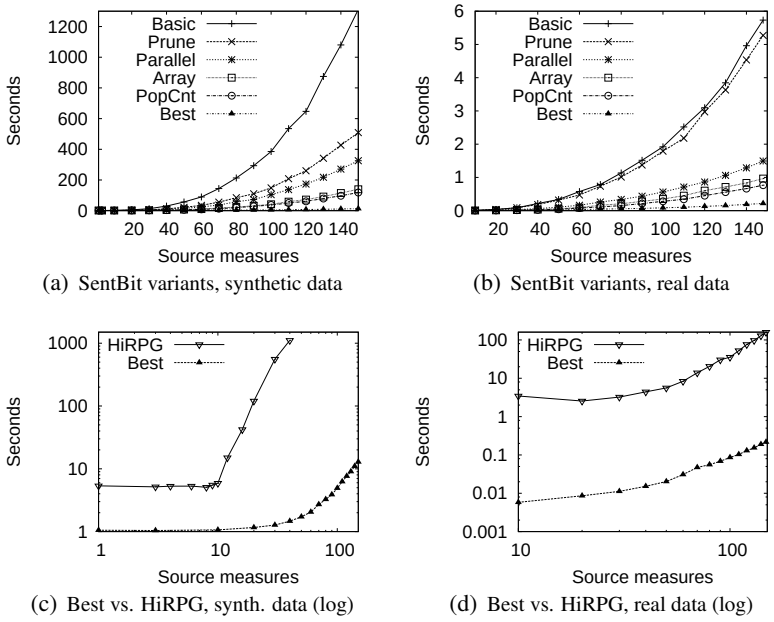
(b) SentBit variants, real data

(c) Best vs. HiRPG, synth. data (log)

(d) Best SentBit, synthetic data (log)

Figure 6.1: Scaling dataset size

that can be observed for all variants in Figure 6.1(a) is caused by minor differences in the randomly generated synthetic datasets. In Figures 6.1(a) and 6.1(b) we observe each optimization option to have effect compared to the "Basic" variant with no optimization. We observe that "Prune" is more efficient on the synthetic data, using only 44% of "Basic" runtime on average (Figure 6.1(a)), compared to the real data where "Prune" used 94% of "Basic" runtime on average (Figure 6.1(b)). This could question our assumption that $\frac{1}{3}$ of a realistic database has no relevant changes. Nevertheless, based on our experience we still maintain that in many cases this occurs, and as observed "Prune" is efficient in both cases. In Figures 6.1(a) and 6.1(b) we see the following performance ranking of the variants: "Basic" (slowest), "Prune", "Parallel", "Array", "PopCnt", and "Best" (fastest). In particular, we see that the "Array" optimization is very close to "PopCnt", yet slightly less efficient. The proximity of all graphs in Figure 6.1(b) is caused by the fact that the encoding part of the algorithm accounts for a larger share of the total algorithm runtime. In Figure 6.1 we observe the "Best" variant outperforming any other variant, and in Figure 6.1(c) we observe that it is 300 times faster, and thus far superior to *SentHiRPG* that represent the state of the art. In Figure 6.1(d) we observe that the linear scalability of "Best" continues to the extreme amount of 10,000,000 periods.



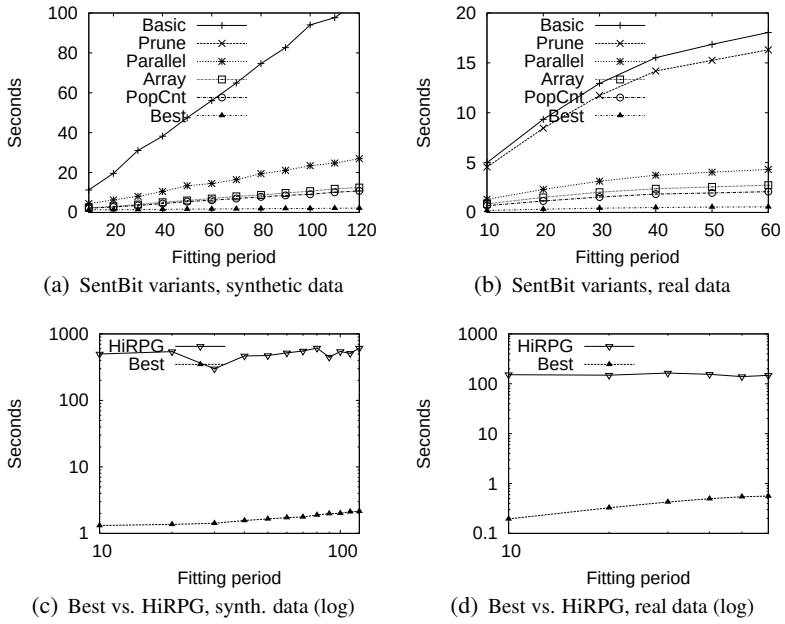
(a) SentBit variants, synthetic data (b) SentBit variants, real data

(c) Best vs. HiRPG, synth. data (log) (d) Best vs. HiRPG, real data (log)

Figure 6.2: Scaling source measures

Scaling source measures: In Figure 6.2 we scale the number of source measures from 1 to 150 on a synthetic dataset and from 10 to 148 on the real dataset. With reference to Section 6.6 we note that increasing the number of source measures is expected to have a cubic impact based on our assessment of the computational complexity and *MaxSource*=3 (when the other factors are kept constant), and we have verified in the statistical tool R (`www.r-project.org`) that the curves in Figure 6.2 are indeed cubic. Similar to scaling the dataset size, we observe that "Prune" is more efficient on the synthetic data (Figure 6.2(a)) than on the real data (Figure 6.2(b)). In addition, we note that the ranking in performance of the variants is similar to that observed in Figure 6.1. We also observe again that "Best" significantly outperforms *SentHiRPG* from prior art.

Scaling the fitting period: In Figure 6.3 we validate that all variants of *SentBit* scales linearly when we vary the *Maxw* over which we fit the warning period, *w* (when the other factors are kept constant). This is particularly evident in Figure 6.3(a) where *Maxw* varies from 10 to 120 periods on a dataset with 100,000 periods. When scaling *Maxw* from 10 to 60 periods on the real dataset in Figure 6.3(b) we see a slightly sub-linear behavior which is due to the dataset only containing 241 periods. Since $\frac{Maxw}{\#Periods}$ is much higher than for the synthetic data, the number of periods that are excluded from the mining process becomes significant in terms of runtime. (We will, however, still characterize the behavior when scaling *Maxw* as linear.) The ranking



(a) SentBit variants, synthetic data

(b) SentBit variants, real data

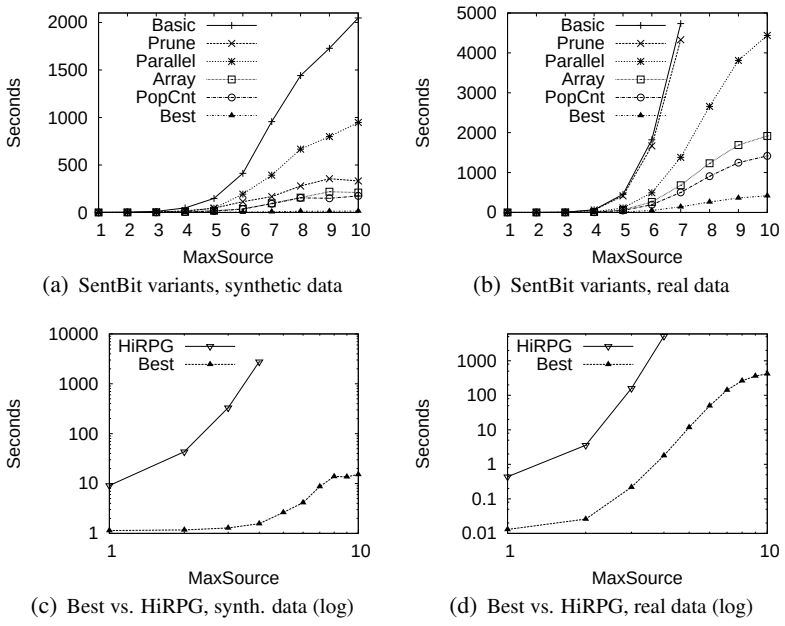
(c) Best vs. HiRPG, synth. data (log)

(d) Best vs. HiRPG, real data (log)

Figure 6.3: Scaling the fitting period

of variants and behavior of "Prune" on synthetic vs. real data is similar to what we observed in the previous experiments. In Figures 6.3(c) and 6.3(d) we observe the significant difference between "Best" and *HiRPG*, where prior art is 300 (synthetic data) – 400 (real data) times slower than the "Best" variant of *SentBit*. The variation in runtime that can be observed on *HiRPG* is due to a random starting point in the hill-climb approximation that this algorithm uses for optimization. The extremely slow gain of *HiRPG* on *SentBit* is due to the usage of hill-climb approximation which we have chosen not to use in *SentBit* to ensure that it also works on non-convex functions.

Scaling *MaxSource*: In Figure 6.4 we scale *MaxSource* for the *SentBit* variants on synthetic (Figure 6.4(a)) and real data (Figure 6.4(b)). As expected, the runtime for all variants initially grows exponentially in *MaxSource*. However, we found that extreme scaling of *MaxSource* behaves differently. The effect is particularly evident in Figures 6.4(a) and 6.4(b) where we observe the "Array", "PopCount", and "Best" variants to "even out" for $MaxSource \geqq 8$. This "broken" exponential growth is due to the weights in *Score* (Formula 6.9) that are effectively preventing overfitting of rules [45], i.e., *Score* for the sentinels with 8 source measures is improving in much fewer cases when adding an additional source measure when $MaxSource \geqq 9$. With



(a) SentBit variants, synthetic data (b) SentBit variants, real data

(c) Best vs. HiRPG, synth. data (log) (d) Best vs. HiRPG, real data (log)

Figure 6.4: Scaling *MaxSource*

prior art (Chapter 3) we never observed this phenomenon since poor performance never permitted us to run experiments for $MaxSource > 6$.

Qualitative Experiment: Apart from assessing the performance of *SentBit*, we also verified it to provide the same sentinels as prior ar (Chapters 2 and 3). Moreover, we found prior art to dismiss better rules where *SentBit* would not. Motivated by review comments on prior papers, we have also tested the relationship between statistical significance in the data and the sentinels found. We created a range of datasets with 100,000 to 1,000,000 periods, where all measure changes were completely random. Therefore there will not exist any statistically significant relationships in the data, and we would not expect *SentBit* to return any sentinels. We verified that *SentBit* did indeed not return any sentinels on these datasets, thus *SentBit* will only find sentinels if statistically significant relationships exist in the data.

Experiments summary: In the experiments we observed all optimizations in Section 6.6 to be efficient, and we found that a best-of-breed approach to create the best combined optimization was applicable, specifically the best *SentBit* algorithm uses *pruning*, *POPCNT*, and *parallelization*. We observed the best *SentBit* algorithm to be 300 – 400 times more efficient than the state of the art. In addition, we found that

SentBit eliminated a weakness in the sentinel mining of prior art that could potentially lead to false positives/negatives. In general, we found *SentBit* to be efficient and to behave as anticipated from our assessment of computational complexity, with the exception of the algorithm behaving better than anticipated when scaling the number of source measures allowed in the sentinels mined due to the characteristics of the test data. In conclusion we can say, that to mine sentinels most efficiently with *SentBit*, pruning should always be applied, POPCNT and parallelization should be used when supported by the CPU available, and "Array" is the best alternative if POPCNT is not available. The selection of optimization can thus be done using a best-of-breed approach based on the optimizations available.

6.8 Conclusion and Future Work

We have showed how to represent the sentinel mining problem by bitmap operations, using *bitmapped encoding* of so-called *indication streams*. We have proposed a very efficient algorithm for bitmapped sentinel mining, *SentBit*, that is 2–3 orders of magnitude faster than the state of the art. Furthermore, *SentBit* does not use approximation, and thus provides exact results unlike prior art. In conjunction with *SentBit*, we provided a number of optimizations utilizing CPU specific instructions and the multi-core architectures available on modern processors. Finally, we have presented experiments demonstrating that *SentBit* scales efficiently to very large datasets, and that sentinels are only found if the data contains statistically significant relationships. We have previously demonstrated that sentinels can find strong and general rules that would not be found by sequential pattern mining or correlation techniques (Chapter 2), this obviously also holds for bitmapped sentinel mining.

For future work, a natural development would be to mine sentinels for multiple target measures in parallel to improve performance. Secondly, we could exploit the multi-dimensional environment by having sentinel mining fit the aggregation level on dimensions as well as select the location and shape of the data area. Third, we would like to induce more flexibility by introducing intervals to replace the fixed warning period and the offset.

Chapter 7

Multidimensional Sentinel Mining Using Bitmaps

This chapter proposes a highly efficient bitmap-based approach for discovery of so-called *multidimensional sentinels*. Sentinels represent *schema level* relationships between *changes over time* in certain measures in a multidimensional data cube. Multidimensional sentinels notify users based on previous observations in subsets of the cube, e.g., that revenue might drop within two months if an increase in customer complaints in USA (drilldown into geography dimension) combined with a decrease in the money invested in customer support for laptop computers (drilldown into product dimension) is observed. We significantly extend prior work by allowing sentinel mining to discover patterns that include the hierarchical dimensions in an OLAP cube. We present a very efficient algorithm, *SentBMD*, for multidimensional sentinel mining using bitmaps, that allows source measures to be progressively input during the mining process, and that parallelizes the work over multiple cores on modern CPUs. The *SentBMD* algorithm is significantly more efficient than prior art, and it scales efficiently to very large datasets, which is verified by extensive experiments on both real and synthetic data.

7.1 Introduction

The Computer Aided Leadership and Management (CALM) concept copes with the challenges facing managers that operate in a world of chaos due to the globalization of commerce and connectivity [36]; in this chaotic world, the ability to continuously *react* is far more crucial for success than the ability to *long-term forecast*. The idea in

CALM is to take the Observation-Orientation-Decision-Action (OODA) loop (originally pioneered by "Top Gun" fighter pilot John Boyd in the 1950s), and integrate business intelligence (BI) technologies to drastically increase the speed with which a user in an organization cycles through the OODA loop. One way to improve the speed from observation to action is to expand the "time-horizon" by providing the user of a BI system with warnings based on "micro-predictions" of changes to an important measure, often called a Key Performance Indicator (KPI). A *sentinel* is a causal relationship where changes in one or *more source measures*, are followed by changes to a *target measure* (typically a KPI), within a given time period, referred to as the *warning period*. We attribute higher quality to *bi-directional* sentinels that can predict changes in both directions, since such a relationship intuitively is less likely to be coincidental (see Section 7.2). An example of a multidimensional sentinel for a computer company could be: "IF number of customer complaints go up in USA and support costs go down in USA for laptop computers THEN revenue goes down within three months AND IF number of customer complaints go down in USA and support costs go up in USA for laptop computers THEN Revenue goes up within three months". Such a rule will allow a BI system to notify a user to take corrective action once there is an occurrence of, e.g., "customer complaints go up in USA and support costs go down in USA for laptop computers", since he knows, based on the "micro-prediction" of the rule, that revenue, with the probability stated by the rule's confidence, will go down in two months if no action is taken. One course of action that would appear to reasonable, based on the sentinel, would be to invest more in the support for laptop computers in USA to address the complaints. In this case it is important to note, that an ordinary sentinel rule might not exist for *all* geographical regions or *all* computer products, and thus the benefit of multidimensional sentinel mining is that we uncover causal relationships that would otherwise go undetected by ordinary sentinel mining. In Section 7.2, we elaborate on the difference between multidimensional and ordinary sentinel mining.

Our contributions are as follows. First, we significantly extend prior work to exploit the hierarchical nature of a multi-dimensional database by allowing sentinels to be discovered for certain dimension values for certain source measures. Second, we present a very efficient algorithm, *SentBMD*, for multidimensional sentinel mining using bitmaps, that allows source measures to be progressively input during the mining process. Third, our *SentBMD* algorithm utilizes new CPU architectures optimally by parallelizing the work over multiple cores and taking advantage of novel CPU instructions. Fourth, we demonstrate that *SentBMD* is more efficient than prior art even when mining ordinary (non-multidimensional) sentinel rules. Fifth, we present experiments demonstrating that *SentBMD* scales efficiently to very large datasets.

Compared to prior art, sentinels are mined on the measures and dimensions of multiple cubes in an OLAP database, as opposed to the "flat file" formats used by

most traditional data mining methods. Sentinels find rules that would be impossible to detect using traditional techniques, since sentinels operate on *data changes* at the *schema level* as opposed to absolute *data values* at the *data level* such as association rules [2] and sequential patterns typically do [5]. In Chapter 2 we specifically provide a concrete, realistic example where nothing useful is found using these techniques, while sentinel mining *do* find meaningful rules. The nature of changes at the schema level gives rise to bitmapped encoding (Section 7.4) of indication streams (Section 7.3), which is the prerequisite of bitmapped sentinel mining, and therefore association rules and sequential pattern mining will not be able to find the same rules. In particular, the bi-directional property of sentinels is significantly different for quantitative applications of these techniques such as [46]. In addition, the auto-fitting of the warning period, and the ability to combine source measures into better sentinel rules, adds to the distance between our solution and the results and optimizations offered in prior art such as [3, 11, 24, 49, 53, 57, 59].

Gradual rule mining [7, 10, 27, 31] is a process much like association rules, where the categorical data are created by mapping numerical data to fuzzy partitions, and thus this technique works on numerical data similar to sentinels. However, similar to association rules and sequential patterns, gradual rule mining does not have the schema level property of sentinels that allows sentinel mining to create the strong bi-directional rules.

Other approaches to interpreting the behavior of data sequences are various regression [4] and correlation [26,60] techniques which attempt to describe a functional relationship between one measure and another. In a multi-dimensional database such regression techniques (Bellwether Analysis) can be used on historical sales data to identify a leading indicator that can predict the global sales of a new product, for which only a short period of data is available [13, 14]. However, similar to gradual rules, these techniques are also concerned with the absolute values of a measure, as opposed to sentinels that are based on changes in the measure values. With regards to the output, sentinels are more *specific "micro-predictions"*, i.e., strong rules that hold for a subset of the data and stimulate specific actions, and are thus complementary to these techniques. Sentinels are therefore useful for detecting incidents and generating warnings whenever changes (that would otherwise go unnoticed) in a relevant source measure occur (Chapter 2).

With regards to parallelization of our bitmapped sentinel mining algorithms, prior art in other aspects of data warehousing have also applied parallelization in order to deal with the huge volumes of data that resides in these systems [17, 18]. In addition, multi-core parallelism has been applied to gradual rule mining [31].

Sentinel mining from a user and an industry perspective has been described in Chapters 4 and 5, and the underlying algorithms have been described in Chapters 2 and 3. Compared to the previously most efficient sentinel mining algorithm described

(a) Source measures filtered by dimension values

Month	*Complaints (All)*	*Complaints (USA)*	*Complaints (Laptop)*	*Support Cost (All)*	*Support Cost (USA,Laptop)*
2009-Jan	726	225	435	750	168
2009-Feb	799 ▲	254 ▲	432	802	165
2009-Mar	720	254	492 ▲	767	135 ▼
2009-Apr	802 ▲	381 ▲	326 ▼	788	101 ▼
2009-May	743	254 ▼	293 ▼	783	363 ▲
2009-Jun	753	381 ▲	378 ▲	768	310 ▼
2009-Jul	653 ▼	254 ▼	331 ▼	779	440 ▲
2009-Aug	675	508 ▲	379 ▲	762	297 ▼
2009-Sep	699	635 ▲	328 ▼	781	260 ▼
2009-Oct	728	699 ▲	366 ▲	769	230 ▼
2009-Nov	668	508 ▼	396	785	294 ▲
2009-Dec	657	635 ▲	468 ▲	801	264 ▼
2010-Jan	818 ▲	762 ▲	445	809	230 ▼
2010-Feb	784	508 ▼	455	780	270 ▲
2010-Mar	706	381 ▼	505 ▲	786	353 ▲
2010-Apr	687	254 ▼	432 ▼	793	400 ▲

(b) Target measure

Month	*Revenue (All)*
2009-Apr	900
2009-May	1001 ▲
2009-Jun	1200 ▲
2009-Jul	750 ▼
2009-Aug	1001 ▲
2009-Sep	1100
2009-Oct	1250 ▲
2009-Nov	970 ▼
2009-Dec	850 ▼
2010-Jan	720 ▼
2010-Feb	1250 ▲
2010-Mar	930 ▼
2010-Apr	800 ▼
2010-May	1100 ▲
2010-Jun	1400 ▲
2010-Jul	1600 ▲

Table 7.1: The relationship between two filtered source and one target measure

in Chapter 6, we have extended sentinel mining with the ability to mine multidimensional data. In comparison, the *SentBMD* algorithm is far more efficient and needs only half of the invocations when conducting the mining process. In addition, *SentBMD* loads and encodes source measure bitmaps in parallel with the mining process and is thus even more efficient.

The remainder of the chapter is structured as follows: The next section intuitively presents the concept of multidimensional sentinels, Section 7.3 presents the formal definition, Section 7.4 presents the prerequisites for multidimensional sentinel mining using bitmaps, Section 7.5 presents the new multidimensional *SentBMD* algorithm, and Section 7.6 presents the optimization and implementation of *SentBMD*, Section 7.7 presents experimental results, and Section 7.8 presents our conclusions and proposals for future work.

7.2 Multidimensional Sentinels

Table 7.1 is a data example for a computer company, where two subsets have been extracted from a database. The data has been organized multidimensionally in three *dimensions*, namely, time, geography, and product. This allows us to easily access the *measures*, complaints, support cost ($1,000s), and revenue ($1,000s), on a monthly level for particular elements in the geography and product dimensions. In Table 7.1, the two source measures have been filtered on different dimensions, i.e., the number of complaints received from customers have been filtered by "USA" on the geographical dimension as well as by "Laptop" on the product dimension, and the support cost

has been filtered by both "USA" and "Laptop" on their respective dimensions. In addition, both source measures have been extracted without any filter (shown as "All"). The source measures in Table 7.1(a) have been extracted for January 2009 to April 2010. The target measure in Table 7.1(b), represents the total revenue for the company, and it has been extracted for April 2009 to July 2010; a similar period in length starting three months later. We refer to these three months as the *Warning Period*. We have calculated the cases where a measure changes 10% or more, either up (▲) or down (▼), from one month to another. We refer to each change to a measure over time as an *indication*, *Ind*.

As seen in the 16 rows in Table 7.1, the measures *Complaints(USA)* and *Support Cost(USA,Laptop)* tend to change in a combined pattern such that when *Complaints(USA)* goes up, *Support Cost(USA,Laptop)* goes down, and vice versa. This source measure pattern is observed 13 times, out of 15 possible. If we combine this pattern with the subsequent changes to *Revenue* three months later, we see that *Revenue* changes in the same direction as *Support Cost(USA,Laptop)* in 12 out of 13 possible times (denoted by $\#ChangesToSource = 13$). Another observation is that the relationship between *Revenue* and the combination of *Complaints(USA)* and *Support Cost(USA,Laptop)* goes in both directions, which is a property we refer to as *bi-directionality*. Intuitively, one can say that if a relationship is bi-directional, then there is a greater chance that the relationship is causal, as opposed to a uni-directional relationship where a pattern is observed for measure changes in one direction only. Consider a case where revenue and staff costs increase over a period of time. This yields the uni-directional relationship that an increase in revenue leads to an increase in staff costs the following month; in this case a decrease in revenue will not necessarily lead to a decrease in staff costs since these costs tend to be more fixed. Therefore, bi-directional relationships are more desirable. It is also noteworthy that *Revenue* changes 6 times up (denoted by $A = 6$) and 6 times down (denoted by $B = 6$) in combination with *Complaints(USA)* and *Support Cost(USA,Laptop)* since this "balance" again adds to the likeliness that the relationship is indeed causal. In summary we can say that a sentinel exists in Table 7.1 where changes in *Complaints(USA)* and *Support Cost(USA,Laptop)* is able to warn three months ahead about changes to *Revenue* with a *Confidence* of 92% (12 out of 13 times), defined as $Confidence = \frac{|A+B|}{\#ChangesToSource}$. $Balance = \frac{4\times|A|\times|B|}{(|A|+|B|)^2}$ is a measure for the degree to which a sentinel is balanced, and in this case the sentinel is perfectly balanced, meaning that $Balance = 1$. We note that the bi-directional quality that can be achieved by assessing *Balance*, is impossible to achieve for sequential patterns since they can only represent one direction of changes in each pattern.

In addition to the combined relationship of the source measures, we can also observe "simple" sentinels (Chapter 2) with only one source and one target measure in Table 7.1. However, the *inverted* relationship between *Complaints(USA)* and

Revenue, as well as the relationship between *Support Cost(USA,Laptop)* and *Revenue*, each have one occurrence (the first two changes) where *Revenue* changes in the opposite direction of what we would expect from all other changes. To assess the prediction ability for such sentinels we must first eliminate its *internal contradictions*. This is done by deducting the number of times *Revenue* changes in the "unexpected" direction from the number of times *Revenue* changes in the "expected" direction. This means that both source measures change 14 times, whereas the target measure after elimination changes only 11 times $(12 - 1)$. Therefore the simple sentinels have a poorer *Confidence* of 79% ($\frac{5+6}{14}$) and are slightly less balanced ($Balance = \frac{4\times|5|\times|6|}{(|5|+|6|)^2} = 0.99$) compared to the sentinel where the source measures were combined. On the other hand, simpler sentinels with fewer source measures have the advantage of being more general than very specific, potentially *overfitted*, sentinels with many source measures, and therefore the simplicity of a sentinel is also important.

In Table 7.1(a), we note that the indications found when applying the filters "USA" and "Laptop" do not translate into similar indications on the "All" level. For example, the reason for support costs being fairly constant at the top level could be that a fixed number of staff is employed, and that the minor fluctuations occur as a result of these employees working overtime. However, when allocating this staff to specific tasks, the cost is allocated accordingly and the fluctuation between tasks will thus generate changes whereof some are indications. Specifically, we note that any ordinary sentinel where *Support Cost(All)* is involved will have no indications on the source measure(s) at all. In addition, a simple ordinary sentinel between *Complaints(All)* and *Renevue(All)* has a *Confidence* of 50% since $A = 1$ and $B = 1$ after elimination of contradictions. Therefore, the potential of multidimensional mining of sentinels for different dimension values is obvious. In Section 7.4, we elaborate further on the issue that different dimension values can have indications that may not be reflected at the upper levels.

7.3 Formal Definition

Let C be a multidimensional *cube* containing a set of *facts*, $C = \{(d_1, d_2, ..., d_n, m_1, m_2, ..., m_p)\}$. The dimension values, $d_1, d_2, ..., d_n$, belong to the *dimensions* $D_1, D_2, ..., D_n$, and we refer to the "dimension part" of a fact, $(d_1, d_2, ..., d_n)$, as a *cell*. We say that a cell belongs to C, denoted by $(d_1, d_2, ..., d_n) \in C$, when a fact $(d_1, d_2, ..., d_n, m_1, m_2, ..., m_p) \in C$ exists. We say that a *measure value*, m_i, is the result of a partial function, $M_i : D_1 \times D_2 \times ... \times D_n \hookrightarrow \Re$, denoted by, $M_i(d_1, d_2, ..., d_n) = m_i$, where $(d_1, d_2, ..., d_n) \in C$ and $1 \leqq i \leqq p$. We refer to M_i as a *measure*. We assume, without loss of generality, that there is only one time dimension, $T \in C$, and that $T = D_1$, and subsequently $t = d_1$. In

addition, we assume that measures $M_1, ..., M_{p-1}$ are source measures, and that measure M_p is the target measure. Finally, we assume that all dimension hierarchies in $D_j \in \{D_1, D_2, ..., D_n\}$ are strict, meaning that each value, d, in a dimension, D_j, has exactly one *parent*, p, with the exception of the "top" ("All") node, $\top_j$, which has no parent. In other words, each dimension is organized as a tree, with a single root. We denote the relationship between a child, $c \in D_j$, and a parent, $p \in D_j$, as $c \sqsubseteq p$. The relationship $\sqsubseteq$ forms a partial order over D_j. We use the same notation for any ancestor, $a \in D_j$, of c, e.g., the parent of a parent, thus if additionally $p \sqsubseteq a$ then $c \sqsubseteq a$.

An *indication*, *Ind*, tells us whether a measure, M_i, changes by a factor of at least α over a period, o. We define $Ind(M_i, d_2, d_3, ..., d_n, t, o)$ when $\{(t, d_2, d_3, ..., d_n),(t+o, d_2, d_3, ..., d_n)\} \subseteq C$ as follows:

$$Ind(M_i, d_2, d_3, ..., d_n, t, o) = \begin{cases} \blacktriangle \text{ if } \frac{M_i(t+o, d_2, d_3, ..., d_n) - M_i(t, d_2, d_3, ..., d_n)}{M_i(t, d_2, d_3, ..., d_n)} \geqq \alpha \\ \blacktriangledown \text{ if } \frac{M_i(t+o, d_2, d_3, ..., d_n) - M_i(t, d_2, d_3, ..., d_n)}{M_i(t, d_2, d_3, ..., d_n)} \leqq -\alpha \end{cases} \tag{7.1}$$

We refer to $\blacktriangle$ as a *positive* indication and to $\blacktriangledown$ as a *negative* indication. We define a wildcard, ?, that can be either $\blacktriangle$ or $\blacktriangledown$. We define the *complement* of an indication as follows: $\overline{\blacktriangle} = \blacktriangledown$ and $\overline{\blacktriangledown} = \blacktriangle$. In addition, we define the inverted measure, $\overline{M_i(x)} = -M_i(x)$, where all the indications $Ind(\overline{M_i}, d_2, d_3, ..., d_n, t, o) = \overline{Ind(M_i, d_2, d_3, ..., d_n, t, o)}$.

Let $(C \bmod T)$ be the cube, C, with the time dimension, T, projected away. A filter, f_{i_k}, is a cell in $(C \bmod T)$. We refer to the single top node, composed of $\top_2, \top_3, ..., \top_n$ for dimensions $D_2, D_3, ..., D_n \in (C \bmod T)$, as $\top = (\top_2, \top_3, ..., \top_n)$. In our example in Table 7.1(a), we have two non-time dimensions, and we use the filters ("All","All"), ("USA","All"), ("All","Laptop"), and ("USA","Laptop") on the geography dimension, D_2, and the product dimension, D_3. We note, that the filter ("All","All") $= \top$, and that filter values can be from any level in any dimension.

We define a *filtered measure*, $f_{i_k} \circ M_i$, as shown in Formula 7.2:

$$\begin{aligned} f_{i_k} \circ M_i(t, d_2, d_3, ..., d_n) = \\ M_i(t, d_2, d_3, ..., d_n) \text{ if } (d_2, d_3, ..., d_n) = f_{i_k}, \\ \text{undef otherwise} \end{aligned} \tag{7.2}$$

Thus a filtered measure is a measure in itself in the above sense, but simply restricted to fewer dimension values. Specifically there can be k different filters for each measure, where $1 \leqq k \leqq \#cells\ in\ (C \bmod T)$. Filtered measures gives us easy access to specific measure values, m_i, for different time dimension values,

$t \in T$, while all the other dimension values $d_2, d_3, ..., d_n$ are constant. We can also pass a regular measure to a function defined for a filtered measure; in this case we let the the "filter-part" default to $\top$, i.e, $M_i = \top \circ M_i$. We extend our notation for parent/child relationships to include filters. In the case where all dimension values in a filter, f_{i_k}, are ancestors of the corresponding dimension values in another filter, f_{i_l}, we say that $f_{i_l} \sqsubseteq f_{i_k}$.

In Table 7.1(a) we see the values for the filtered source measures *Complaints(USA)*, *Support Cost(USA,Laptop)*, and *Complaints(Laptop)* over different time periods. In this notation, we only write the filters that differ from $\top_j$ on a given dimension, and the dimension (time, geography, or product) is given implicitly by the value. Similarly, in Table 7.1(b) we see the values for the target measure, *Revenue*, but with no filters. In addition to the measure values in Table 7.1, we have listed the indications next to the last value that is part of a measure change, e.g., in Table 7.1(a), column 2, row 2, we see that $Ind(Complaints, USA, 2009 - Jan, 1month) = \blacktriangle$ (we use the first period in an indication, t, to uniquely reference it on the time dimension).

We define an *indication sequence*, *IndSeq*, for a filtered measure, $f_{i_k} \circ M_i$, where $M_i \in \{M_1, ..., M_p, \overline{M_1}, ..., \overline{M_p}\}$, and $f_{i_k} \in (C \bmod T)$ as shown in Formula 7.3. Intuitively, an *IndSeq* captures the non-neutral indications given by the time dimension values on a filtered measure, $f_{i_k} \circ M_i$ for the cells given by its filter, f_{i_k}. In the following we use set notation for convenience, but implicitly assume that the *order* of the sequence is given by the dimension values $t \in T$.

$$
\begin{aligned}
&\mathit{IndSeq}(f_{i_k} \circ M_i, C, o, w) = \\
&\{(t, Ind(M_i, t+w, o, d_2, d_3, ..., d_n)) \mid \\
&\quad \{(t+w, d_2, d_3, ..., d_n), (t+w+o, d_2, d_3, ..., d_n)\} \subseteq C \\
&\qquad \wedge\ (d_2, d_3, ..., d_n) = f_{i_k} \\
&\qquad \wedge\ Ind(M_i, t+w, o, d_2, d_3, ..., d_n) \in \{\blacktriangle, \blacktriangledown\}\}
\end{aligned}
\tag{7.3}
$$

In Table 7.1, we intuitively see the indication sequences as the columns consisting of ▲ and ▼ combined with the respective period in the *Month* column. A join of two or more indication sequences, *IndJoin*, is a set of filtered measures, $\{f_{1_k} \circ S_1, ..., f_{m_k} \circ S_m\} \subseteq \{f_{1_k} \circ M_1, ..., f_{p_k} \circ M_p, f_{1_k} \circ \overline{M_1}, ..., f_{p_k} \circ \overline{M_p}\}$ as shown in Formula 7.4.

$$
\begin{aligned}
&\mathit{IndJoin}((f_{1_k} \circ S_1[, w_1]), ..., (f_{m_k} \circ S_m[, w_m]), C, o[, F]) = \\
&\quad \{(t, \mathit{Ind}(S_1, t + w_1, o, d_2, d_3, ..., d_n)) \mid \\
&\qquad (t, \mathit{Ind}(S_1, t + w_1, o, d_2, d_3, ..., d_n)) \in \\
&\qquad\quad \mathit{IndSeq}(f_{1_j} \circ S_1, C, o, w_1) \wedge \\
&\qquad \mathit{Ind}(S_1, t + w_1, o, d_2, d_3, ..., d_n) = F \wedge \\
&\qquad \forall\, (fS, w) \in \{(f_{2_k} \circ S_2, w_2), ..., (f_{m_k} \circ S_m, w_m)\} : \\
&\qquad\quad (t, \mathit{Ind}(S_1, t + w_1, o, d_2, d_3, ..., d_n)) \in \\
&\qquad\quad\quad \mathit{IndSeq}(fS, C, o, w)\}
\end{aligned}
\tag{7.4}
$$

Formula 7.4 allows optional differences in time, $w_1, ..., w_m \in \mathbb{N}_0$, between the indication sequences joined, referred to as warning periods. Intuitively, ***IndJoin*** is an ***m-way semi-join*** where we take the left indication sequence, $(t, Ind(S_1, t + w_1, o, d_2, d_3, ..., d_n))$, and filter out all indications that do not fit with the other sequences. For source measures it is typical to combine measures for the same time instance, and for target measures for a later time instance (default $w_x = 0$ for all $w_x | 1 \leqq x \leqq m$). In addition, we define an optional ***indication filter***, $F \in \{\blacktriangle, \blacktriangledown, ?\}$, that allows the resulting indication sequence to consist of indications in one direction only (default $F =?$). In general, we use [ ] to denote optional parameters in the equations, and we note that if an optional parameter is not passed then the parameter has the default value stated above.

With these definitions, we can output all sentinels in the cube, *C*, with the offset, *o*, as shown in Formula 7.5. Each sentinel is represented as a set of filtered source measures, *fSource*, a target measure, *Target*, and a warning period, *w*. We use *MaxSource* as a threshold for the maximum number of filtered source measures we want to combine in a sentinel, and we use *Maxw* as a threshold for the maximum warning period we are willing to accept. The thresholds α, β, γ, and σ are global and are thus not passed to Formula 7.5.

$$
\begin{aligned}
SentRules(C, o) = \{(&fSource_l \rightsquigarrow Target_r, w_q) \mid \\
&fSource_l \subseteq \\
&\{f_{1_k} \circ M_1, ..., f_{p-1_k} \circ M_{p-1}, f_{1_k} \circ \overline{M_1}, ..., f_{p-1_k}\overline{M_{p-1}}\} \wedge \\
&\forall\{f_a \circ S, f_c \circ S\} \subseteq fSource_l : (f_c \not\sqsubset f_a) \wedge \\
&|fSource_l| \leqq MaxSource \wedge Target_r \in \{M_p, \overline{M_p}\} \wedge \\
&1 \leqq w_q \leqq Maxw \wedge \\
&SentSupp(fSource_l, C, o) \geqq \sigma \wedge \\
&ElimSupp(fSource_l, Target_r, C, o, w_q, \blacktriangle) \geqq 0 \wedge \\
&ElimSupp(fSource_l, Target_r, C, o, w_q, \blacktriangledown) \geqq 0 \wedge \\
&Balance(fSource_l, Target_r, C, o, w_q) \geqq \beta \wedge \\
&Confidence(fSource_l, Target_r, C, o, w_q) \geqq \gamma\}
\end{aligned}
\tag{7.5}
$$

In Formula 7.5, we exclude parents to filtered source measures that are already "participating" in a sentinel rule. This is done to eliminate redundancy in the information of a rule, since a parent filter does not contribute with any new information that is describing the target measure in more detail (it only indicates that the child behaves similar to its parent). For example, "Europe" is a parent to "Denmark" in the geography dimension, and if a source measure is already filtered on "Denmark", then adding the same source measure with "Europe" as filter does not alter the restrictions on the target measure in any way.

The functions *SentSupp* (Formula 7.6), *ElimSupp* (Formula 7.7), *Balance* (Formula 7.8), and *Confidence* (Formula 7.9) used in $SentRules(C, o)$ are defined as follows:

$$
SentSupp(fSource, C, o) = |IndJoin(fSource, C, o)| \tag{7.6}
$$

$$
\begin{aligned}
ElimSupp(&fSource, Target, C, o, w, F) = \\
&|IndJoin(fSource, (Target, w), C, o, F)| \\
&- |IndJoin(fSource, (\overline{Target}, w), C, o, F)|
\end{aligned}
\tag{7.7}
$$

$$
Balance(fSource, Target, C, o, w) = \frac{4 \times |A| \times |B|}{(|A| + |B|)^2} \tag{7.8}
$$

$$
\begin{aligned}
&Confidence(fSource, Target, C, o, w) = \\
&\quad \frac{|A| + |B|}{SentSupp(fSource, C, o)}
\end{aligned}
\tag{7.9}
$$

where

$$
\begin{aligned}
A &= ElimSupp(fSource, Target, C, o, w, \blacktriangle) \\
B &= ElimSupp(fSource, Target, C, o, w, \blacktriangledown)
\end{aligned}
$$

Formulae 7.6 to 7.9 use the cardinality of indication sequences that result from joining indication sequences based on the measures in *fSource* and *Target*. $A = ElimSupp(fSource, Target, C, o, w, \blacktriangle)$ (Formulae 7.8 and 7.9) is the number of times that *fSource* changes in a certain direction and *Target* subsequently changes in the

Sentinels found ($w = 3$)	*Rule Len*	*SentS.* (*ElimS.*)	*Conf*	*Balance*	*Score*	*OK* ?
$Complaints(USA)$ $\wedge\ inv(Support\ Cost(USA, Laptop)) \rightsquigarrow inv(Revenue)$	2	13 (12)	92%	1	0.71	OK
$Complaints(USA) \rightsquigarrow inv(Revenue)$	1	14 (11)	79%	0.99	0.66	OK
$Support\ Cost(USA, Laptop) \rightsquigarrow Revenue$	1	14 (11)	79%	0.99	0.66	OK
$Complaints(USA)$ $\wedge\ Complaints(Laptop) \rightsquigarrow inv(Revenue)$	2	7 (6)	86%	1	0.33	OK
$Complaints(USA) \wedge Complaints(Laptop)$ $\wedge\ inv(Support\ Cost(USA, Laptop)) \rightsquigarrow inv(Revenue)$	3	7 (6)	86%	1	0.26	OK
$SupportCost(USA, Laptop)$ $\wedge inv(Complaints(Laptop)) \rightsquigarrow Rev$	2	8 (5)	63%	0.96	0.19	Failed
$Complaints(All) \rightsquigarrow inv(Rev)$	1	4 (2)	50%	1	0.08	Failed
$Complaints(Laptop) \rightsquigarrow inv(Rev)$	1	11 (2)	18%	1	0.03	Failed

Table 7.2: Sentinels ordered by their respective *Score*.

"expected" positive (▲) direction, minus the number of times where *Target* changes in the opposite direction. $ElimSupp(fSource, Target, C, o, w, \blacktriangledown)$, denoted by *B*, is calculated in the same way, but for the changes to *Target* in the opposite direction of *A* (▼). We refer to this as the *contradiction elimination process*, where we essentially force a sentinel to be either $fSource \rightsquigarrow Target$ or $fSource \rightsquigarrow \overline{Target}$, and thereby we effectively eliminate both *contradicting* (same premise but different consequent) and *orthogonal* (different premise but same consequent) indications in the sentinel we are evaluating.

Formulae 7.6, 7.8, and 7.9 represent desired qualities of the sentinel: *SentSupp* (Formula 7.6) tells us how often the premise of the sentinel occurs. *Balance* (Formula 7.8) is used to determine the degree to which a generalized sentinel rule is uni-directional (*Balance=0*) or completely bi-directional (*Balance=1*), meaning that there are exactly the same amounts of positive and negative indications on the target measure in the data. *Confidence* (Formula 7.9) tells us the fraction of occurrences where the premise occurs, and the consequent occurs within w time. We denote the minimum threshold for *SentSupp* by σ, the minimum threshold for *Confidence* is denoted by γ, and the minimum threshold for *Balance* is denoted by β.

Aside from the individual quality measures for a sentinel in Formulae 7.6 to 7.9, it is also desirable to have a quality measure that incorporates all these measures into one value; this is relevant if we want to compare multiple different sentinels to identify the best sentinel(s). For this purpose, we define *Score* for a sentinel, $(fSource \rightsquigarrow Target, w) \in SentRules(C, o)$, as shown in Formula 7.10.

$$
\begin{aligned}
&Score(fSource, Target, w, C, o) = \\
&\quad (1 - wp + \frac{(1 + Maxw - w) \times wp}{Maxw}) \\
&\quad \times (\frac{1}{2} + \frac{1 + MaxSource - |fSource|}{MaxSource \times 2}) \\
&\quad \times \frac{ElimSupp(fSource, Target, C, o, w, ?)}{MaxElimSupp(C, o, w)} \\
&\quad \times Confidence(fSource, Target, C, o, w) \\
&\quad \times (\frac{1}{2} + \frac{Balance(fSource, Target, C, o, w)}{2})
\end{aligned}
\tag{7.10}
$$

With this definition of *Score*, we denote the maximal value of $ElimSupp(fSource, Target, C, o, w, ?)$ for any sentinel, $(fSource \rightsquigarrow Target, w) \in SentRules(C, o)$, by $MaxElimSupp(C, o)$. The constant, *wp*, represents the warning penalty, i.e., the degree to which we want to penalize rules with a higher *w* (0=no penalty, 1=full penalty). Generally, *Score* incorporates a preference having high values for all quality measures (Formulae 7.6 to 7.9), and having shorter sentinel rules (low $|fSource|$) with a short warning period, w. The construction of Score is further elaborated in Chapters 3 and 5.

We can now use the *SentRulesPruned* function, as shown in Formula 7.11, where *fSource* and w for a sentinel, $S \in SentRules(C, o)$, are denoted by $S_{fSource}$ and S_w. Formula 7.11 eliminates rules with poor quality (lower *Score*) if a shorter rule exists with at least as good a *Score*, and where *fSource* is a proper subset of *fSource* for the longer rule. In addition, *SentRulesPruned* eliminates rules with similar source measures, but longer warning periods, *w*.

$$
\begin{aligned}
&SentRulesPruned(C, o) = \\
&\quad \{S \in SentRules(C, o) \mid \nexists S' \in SentRules(C, o) : \\
&\qquad (Score(S', C, o) \geqq Score(S, C, o) \wedge \\
&\qquad (S'_{fSource} \subset S_{fSource} \vee \\
&\qquad (S'_{fSource} = S_{fSource} \wedge S'_w < S_w)))\}
\end{aligned}
\tag{7.11}
$$

The output of $SentRulesPruned(C, o, Optimalw(C, o))$ ordered by their respective *Score* are the best sentinels in a database, *C*, with the offset, o. In order not to increase the number of sentinels during the mining process and in the output, we preserve the order 1 to p for all source measures, 1 to n for the dimensions used as filters, and 1 to j for the specific filter values. We only show specific filters in the output when a filter value is different from $\top j$. In addition, we use the following notation when describing a sentinel, $(fSource \rightsquigarrow Target, w) \in SentRulesPruned(C, o)$: we write all source measures in *fSource* with filters in parenthesis and *Target* as one string where the source measures are separated with $\wedge$ and the target measure sepa-

rated with $\rightsquigarrow$, e.g., $A(fA) \wedge B(fB) \rightsquigarrow C$. In the output it should also be specified which warning period, w, that applies to each sentinel.

If we apply Formulae 7.5 to 7.11 to our data example in Table 7.1, and use the notation above, we get the conforming sentinels ("OK") in the first column of Table 7.2 as output when the thresholds for number of source measures, *SentSupp*, *Balance*, and *Confidence* are set as follows: $MaxSource = 3$, $\sigma = 5$, $\beta = 0.8$, and $\gamma = 75\%$. Furthermore, we use $\alpha = 10\%$ and $wp = \frac{1}{2}$. We note that all sentinels are found with a warning period, $w = 3$ (3 months), since this was the only period we mined in our example, however, in a real dataset there will most likely be more warning periods. We should also note in Table 7.2, row 5, that the sentinel *cannot* be simplified into $Complaints(USA, Laptop) \wedge inv(Support\ Cost(USA, Laptop)) \rightsquigarrow inv(Revenue)$ since the indication join of the indication sequences based on *Complaints(USA)* and *Complaints(Laptop)* are not the same as the indication sequence based on *Complaints(USA,Laptop)*. Therefore, Table 7.2 represents the most simplified output possible, given our data example in Table 7.1.

7.4 Prerequisites for SentBMD

In prior art (Chapter 6) we have demonstrated how an indication sequence based on a source or target measure, *IndSeq*, can be encoded into a bitmap consisting of an even number of CPU words. This encoding also holds for a bitmap based on a filtered measure, and thus applies to all measures used in multidimensional sentinel mining using bitmaps. For example, the indication sequence based on the filtered measure *Complaints(USA)* from Table 7.1(a), formally $IndSeq(Complaints(USA), C, 1, 0)$, can be encoded as follows:

Bitmap($Complaints(USA)$**)** aligned in two 32 bit words

	bits set for positive *Ind*	fill of 0 bits to align with word length
PI :	10101011 1011000	0 00000000 00000000
NI :	00010100 0100111	0 00000000 00000000
	bits set for negative *Ind*	fill of 0 bits to align with word length

As shown in *Bitmap(Complaints(USA))*, the bits are set in accordance with the positive indications, *PI*, and the negative indications, *NI*. The position of the indications are given by the order of the values on the time dimension, $t \in T$, thus even if there is no indication (positive or negative), $Ind(Complaints(USA), t, o)$, for a given t, the position will still be occupied by two unset bits for PI and NI, respectively. In our example, there is only a total of 15 possible indications which easily fits into the 32 bit word. In practice, however, PI and NI will most likely need multiple words allocated to fit the number of bits equivalent to the number of $t \in T$.

In Chapter 6, we have shown that the entire sentinel mining process can be conducted on bitmap-encoded measures using a series of AND operations with subsequent counting of the bits set. We refer to the AND operation of two bitmaps as *BitAND*, and the count of bits set in a bitmap as *BitCount*. We have presented the *SentBit* algorithm, that is a highly efficient way to conduct the sentinel mining process on a simple dataset where a number of measures are combined with only one time dimension. In the *SentBMD* algorithm we apply the lessons learned in *SentBit* to a true multi-dimensional environment where sentinels can combined from the measure changes on different dimension levels. In addition, we have been able to reduce the number of invocations needed in the mining process, so we gain efficiency with *SentBMD* even on a non-multidimensional dataset when compared to *SentBit*. *SentBMD* is also structured in a way that allows the measures to be encoded in parallel with the sentinels being mined.

In [55] it has been demonstrated that a greedy top-down approach will fail when seeking for "interesting" dimension values, defined as those values that best explain their parent. In Figure 7.1 we present a similar example related to the data in Table 7.1(a), where the source measure *Complaints* for January and February 2009 has been filtered at three different levels on the geography dimension, namely: *All* (top level), *Region*, and *Country*. The three indications resulting from a change from January to February that is greater than 10% have been illustrated with ▲. As it can be seen, the change in number of complaints in *All* is highly influenced by two changes in USA (38% of the total change) and in Denmark (41% of the total change). How-

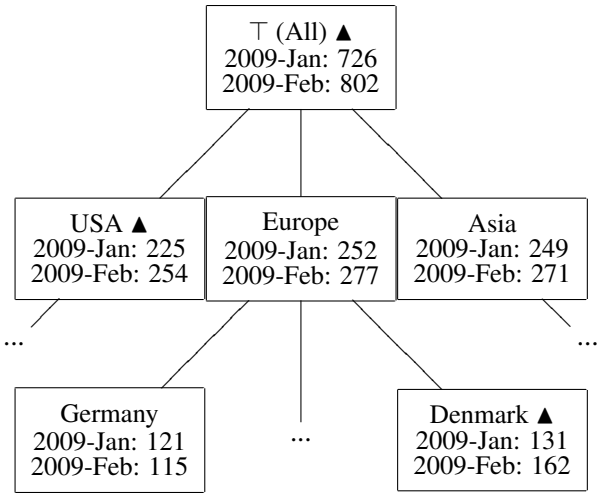


Figure 7.1: Complaints filtered on the geography dimension

ever, even though Denmark accounts for the biggest part of the change in $\top$ overall, its parent, Europe, does not have an indication due to changes for its other children. With this example, we demonstrate an inherent weakness of a greedy top-down approach when seeking for the filters that result in a measure having a certain number of indications that result in a good *Score*. However, a greedy top-down approach is appealing from an efficiency standpoint. In the *SentBMD* algorithm, we have chosen to focus our efforts on an efficient bottom-up approach that finds all sentinels, and we have thus suggested that a top-down approximation would be a natural next step for future work (see Section 7.8).

With these prerequisites we can say, it will be possible to encode all filtered source measures similar to regular source measures, and thus we can mine multidimensional sentinels using bitmaps. In the next section we present the *SentBMD* algorithm that is constructed using these guidelines.

7.5 The SentBMD Algorithm

The motivation for the *SentBMD* algorithm is obviously to be able to mine multidimensional sentinels that cannot be mined with prior algorithms such as *SentBit*. However, the fact that the workload in terms of the number of source measures is much higher when mining multidimensional sentinels has also motivated us to revisit the general approach to sentinel mining that was used in *SentBit*. In *SentBit*, the entire number of source measures had to be encoded into bitmaps before the data could be tested for the existence of sentinels. The disadvantage of this approach is particularly obvious in the case where the encoding process accounts for half of the workload (or more), as shown in Figure 7.2. In Figure 7.2, we show the workload distribution over time for the *SentBit* and the *SentBMD* algorithms on a CPU with four cores. The workload of encoding (E) and the workload of testing (growing) sentinels (G) has been divided into eight equal slices. Since *SentBit* does not have the ability to test the sentinels while the measures are progressively encoded, we need to encode all measures before we can start testing the bitmaps for sentinels. Moreover, *SentBit* did not have the ability to encode in parallel either. With *SentBMD*, the measures can

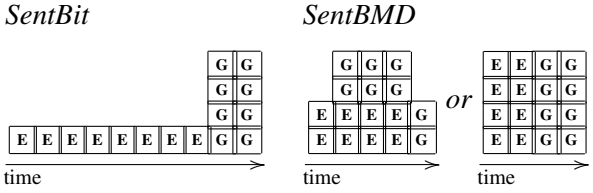


Figure 7.2: Workload distribution for *SentBit* and *SentBMD*

be progressively fed to the process that tests the sentinels and the encoding process can run in parallel, and thus we are able to significantly reduce the time needed for the entire sentinel mining process even when we are mining non-multidimensional sentinels. Although *SentBMD* supports both scenarios in Figure 7.2, it should be noted that the scenario to the right in Figure 7.2 is not realistic in most cases since the encoding process will be constrained by access to either RAM or disk, and thus the middle parallelization strategy has experimentally proven to be the best.

In addition to offering the parallelization strategies shown in Figure 7.2, the *SentBMD* algorithm has also been restructured such that the testing of sentinels can be done in fewer process invocations than with *SentBit*.
The *SentBMD* algorithm initiates two types of processes that run in parallel, *EncodeBlock* that encodes all source measures filtered by dimension values from one or more dimensions, and the recursive *GrowSentinel* function that *grows* the potential sentinels in terms of the number of filtered source measures. The *SentBMD* algorithm divides the total set of filtered source measures into blocks and spawns multiple invocations of *EncodeBlock* in which the members of these blocks are encoded into bitmaps in parallel. Upon completion of the encoding of the first block, multiple invocations of *GrowSentinel* will test the combinations of filtered source measures that qualify to participate in good sentinels. Each invocation of *GrowSentinel* will process the encoded filtered source measures from the blocks available. Upon completion of a block where there are sentinels that can be used for growing longer and higher scoring sentinels, the state of *GrowSentinel* is put in a queue until more blocks of filtered source measures become available. When the last *EncodeBlock*, and subsequently all instances of *GrowSentinel*, complete, the best sentinels can be output from memory. Multidimensional sentinel mining using bitmaps (the *SentBMD* algorithm) is illustrated in Figure 7.3.

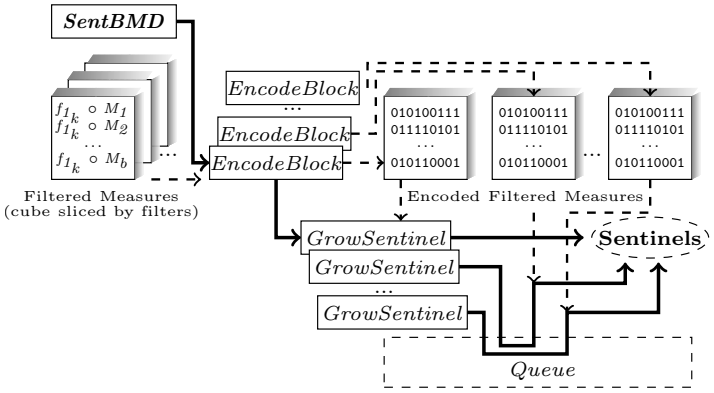


Figure 7.3: The parallel processes in the *SentBMD* algorithm

In *SentBMD* (Algorithm 7.1) and its sub-algorithms, the following dynamic arrays are globally available to all sub-algorithms in order to store the mining results as the recursion progresses: *SentList* is a dynamic array of (a set of measures *Source*, measure *Target*, *w*, *NewScore*) and *SourceList* is a dynamic array that is used to reference filtered source measures. In addition, all bitmaps for encoded measures and the value *MaxElimSupp* (Default 0) are also global. Each filtered source measure in *SourceList* can be referenced by a *lookup* into an array, i.e, *SourceList[x]* where x is the filtered measure number. The variable *BlockNbr* and the constant *BlockSize* are also global. Since *SentList*, *SourceList*, *MaxElimSupp*, and *BlockNbr* can be updated by multiple parallel invocations, we apply a semaphore to control access to the structures.

Upon receipt of the desired number of sentinels, n, the dataset, C, the source and target measures, the offset, o, and the thresholds, the algorithm initiates the mining process by encoding the target measures into bitmaps (line 1). In line 2, *SentBMD* allocates memory for the output of the mining process and the filtered source measure lookup (In Section 7.7, we present results that allows us to estimate the expected memory needed). In lines 3 to 7, filtered source measures to be encoded are added to the *SourceList* structure, i.e., all source measures filtered by non-time dimension values that have no more ancestors than *HierarchyDepth* on the path to the root, $\top_j$. It is important to note that the order in which filtered source measures are encoded and passed to the *GrowSentinel* function (Algorithm 7.4) is irrelevant for the output of *SentBMD* given the AND operation to which the filtered measures are subjected. However, we ensure that the different measures are spread as much as possible by having the loop with the source measures at the bottom (line 6). This reduces the chance that *GetMeasures* will return a large sequence where most filtered measures are based on the same measure, and thus have a higher chance of being excluded by *GrowSentinel* due to their parent relationship (Algorithm 7.4, line 15). We note from our example in Figure 7.1, that setting $HierarchyDepth = 1$ would mean, that only the values "All", "USA", "Europe", and "Asia" would be used as filters on the geography dimension. Similarly, setting $HierarchyDepth = 0$ would mean that we get no multi-dimensionality (same as no filters) in our sentinel mining process, and its results will thus be similar to those produced with the *SentBit* algorithm from Chapter 6. Given the number of members in *SourceList*, we spawn a number of invocations of *EncodeBlock* (Algorithm 7.2) in accordance with the *BlockSize* constant (lines 8 and 9). Once a given source measure is encoded by *EncodeBlock* (Algorithm 7.2), its bitmap is allocated and thus globally available to all other sub-algorithms.

In *SentBMD* (Algorithm 7.1) line 10 we prepare to test the sentinels by setting the first block to be tested, *BlockNbr*, to zero as well as setting the queue for *GrowSentinel* states to empty. In *SentBMD*, line 11, we use the *GetMeasures* sub-function

Algorithm 7.1 *SentBMD*: Multi-dim. sentinel mining w/bitmaps

SentBMD(the max number of sentinels to be returned, n,
a dataset, C,
a set of source measures, $\{S_1, ..., S_p\} \subseteq C$,
a target measure, $Target \in C$,
the *BlockSize*,
an offset, o,
the thresholds, α, β, γ, σ, *Maxw*, *MaxSource*, and *HierarchyDepth*)

1: Encode $Target_w$ for all $w \in \{1, ..., Maxw\}$
2: Allocate memory for *SentList* and *SourceList*
3: **for all** $D \in C$ where $D \neq T$ **do**
4: **for all** ancestors, $a \in D$, where number of ancestors above a are less than or equal to *HierarchyDepth* **do**
5: **for all** $d \sqsubseteq a$ **do**
6: **for all** $S \in \{S_1, ..., S_p\}$ **do**
7: $SourceList \leftarrow SourceList \cup S(d)$
8: **for all** $b \in \{0, ..., \lfloor \frac{|SourceList|}{BlockSize} \rfloor\}$ **do**
9: Spawn *EncodeBlock(b,BlockSize)*
10: $SentQ = \emptyset$
11: **for all** $BlockNbr \in \{0, ..., \lfloor \frac{|SourceList|}{BlockSize} \rfloor\}$ where $GetMeasures(BlockNbr) \neq \emptyset$ **do**
12: **for all** $AddSource \in DataBlock$ **do**
13: $SentQ \leftarrow SentQ \cup (AddSource, \emptyset, Bitmap_{AllBitsSet}, Target, \{1, ..., Maxw\}, 0)$
14: **for all** $Q \in SentQ : (x < BlockNbr \times BlockSize)$ **do**
15: Spawn $GrowSentinel(Q)$
16: $SentQ \leftarrow SentQ \bmod Q$
17: **return** the top n sentinels from *SentList* in the output $Score \leftarrow \frac{NewScore}{MaxElimSupp}$

Algorithm 7.2 *EncodeBlock*: Encode a block of source measures

EncodeBlock(The block number to be encoded, b)

1: **for all** $S \in \{SourceList[b \times BlockSize], ..., SourceList[((b + 1) \times BlockSize) - 1]\}$ **do**
2: Encode *S* into bitmap

(Algorithm 7.3) to read the blocks of filtered source measures as they become available.

Algorithm 7.3 *GetMeasures*: Gets a filtered source measure block

```
    GetMeasures(the block number to be read, b)
 1: DataBlock ← ∅, x ← 0, successful ← true
 2: while x < BlockSize ∧ successful do
 3:    a ← (b × BlockSize) + x
 4:    if SourceList[a] is allocated then
 5:       DataBlock ← DataBlock ∪ a
 6:    else
 7:       if EncodeBlock still running then
 8:          Wait for SourceList[a] to be allocated
 9:          DataBlock ← DataBlock ∪ a
10:       else
11:          successful ← false
12:    x ← x + 1
13: return DataBlock
```

In *GetMeasures* (Algorithm 7.3) we simply test whether the bitmaps for all filtered source measures in a block are allocated, if this is not the case, and an invocation of *EncodeBlock* is still running, then *GetMeasures* will wait for the respective bitmap to be allocated (line 8). If a bitmap is not allocated and no invocation of *EncodeBlock* is running, then there are no more bitmaps left to encode, and *GetMeasure* will terminate unsuccessfully (line 11) with a *DataBlock* that has less members than *BlockSize*. This will only happen for the last block that is encoded.
In *SentBMD* (Algorithm 7.1), newly available blocks of filtered source measures are processed as they become available by intuitively adding a) the filtered source measure to a sentinel with b) no previous source measures, c) a bitmap with all bits set, d) the target measure, and e) a $Score = 0$ to the queue, *SentQ* (Algorithm 7.1, lines 12 to 13). In addition to the new members added to *SentQ* with the receipt of a new *DataBlock*, *SentQ* also holds queued invocations of *GrowSentinel* that were previously halted due to a lack of bitmap encoded measures (see detailed description of *GrowSentinel* below). In lines 14 to 16, all new and halted members in *SentQ* are each used to spawn an invocation of *GrowSentinel*, and subsequently removed from the queue. The process of spawning *GrowSentinel* from *SentQ* is repeated (lines 11 to 16) until all filtered source measures have been encoded and attempted "grown" (The parallelism in the mining process is illustrated in Figure 7.3). Upon completion of the recursive and parallel mining process, the process of outputting maximum n sentinels found (Formula 7.11), sorted descending by *Score*, is trivial (line 17). We note that the outstanding division of *NewScore* by $MaxElimSupp$ in accordance with

Formula 7.10 is done only on the maximum n sentinels output (see explanation of the *CalcScore* function, Algorithm 7.5).

The *GrowSentinel* function (Algorithm 7.4) tests if adding the filtered source measure, *SourceList[AddSource]*, to a given set of filtered source measures, *Source*, would improve the sentinel relationship with the target measure, *Target*. Since *GrowSentinel* is recursive, we potentially need the bitmap representing the filtered source measures many times during the recursion, and to optimize this we apply a standard dynamic programming approach by passing the bitmap, *Bits*, representing the AND operation of all bitmaps for the measures in *Source*. In addition, the function receives the *Score* for the sentinel represented by *Source* and *Target* without the added *SourceList[AddSource]*.

In *GrowSentinel*, there are two options for adding the additional filtered source measure, namely directly, $SourceList[AddSource]$, or inverted, $\overline{SourceList[AddSource]}$, as seen in line 1. The bitmap for the added source measure is joined (AND'ed) with the bitmap for the existing source measures, *Bits*, in line 2, and the result is checked to see if it has less bits set than before in line 3 (the *BitAND* and *BitCount* functions are described in Section 7.4). If the *BitAND* does *not* result in *fewer* bits, then the added source measure is exactly equal to the source measures that are already in the sentinel, meaning that an equally good sentinel can be created with the new source measure alone (this sentinel will be discovered in a later invocation), and thus it is irrelevant to continue with the longer sentinel at hand. We note that sending a bitmap, *Bits*, will all bits set (both PI and NI) will satisfy the condition in line 3 for the any source measure during the initial invocation call from *SentBMD* (Algorithm 7.1, line 13). If the new source measure did indeed reduce the number of bits, we add it to the set of filtered source measures representing a new potential sentinel (line 4). In addition, we set the collection of warning periods where the added source measure improved the *Score* to $\emptyset$.

We can now test if the reduced number of bits translates into a better *Score* for either $Target$ or $\overline{Target}$ for any w where $1 \leqq w \leqq Maxw$ (lines 5 to 6), and we note that the *CalcScore* function (Algorithm 7.5) will return which version of the target measure has the highest score along with *NewScore*. In this regard, we note that there is a good chance that a sentinel will have the same direction on the target measure even though it is made more specific by adding an additional filtered source measure, but we cannot know this for sure, e.g., if adding the filtered source measure reduces the number of indications covered the sentinel by more than 50%. If *NewScore* for the new potential sentinel is better than *Score* from its ancestor (line 7), we record the sentinel in *SentList* if it meets the defined thresholds (lines 8 and 9), and we also update $MaxElimSupp$ if necessary (lines 10 and 11). If the sentinel has less than *MaxSource* filtered source measures and it improved *Score* for at least one warning period, w, (line 13) it is a candidate for growing a longer sentinel with a

Algorithm 7.4 *GrowSentinel*: Add a filtered measure to a sentinel

GrowSentinel(filtered source measure number *AddSource*,
set of filtered measures *Source*, bitmap *Bits*,
a measure *Target*, a warning period set W, *Score*)

1: **for all** $S \in \{SourceList[AddSource], \overline{SourceList[AddSource]}\}$ **do**
2: $NewBits \leftarrow BitAND(Bitmap(S), Bits)$
3: **if** *BitCount(NewBits,{PI,NI})<BitCount(Bits,{PI,NI})* **then**
4: $NewSource \leftarrow Source \cup S$, $NewW = \emptyset$
5: **for all** $w \in W$ **do**
6: $CalcScore(BitAND(NewBits, Bitmap(w)_{AllBitsSet}), BitAND(Bitmap(Target_w), NewBits), BitAND(Bitmap(\overline{Target_w}), NewBits), Target, |NewSource|, w)$
7: **if** *NewScore>Score* **then**
8: **if** $SentSupp \geqq \sigma \wedge Confidence \geqq \gamma \wedge Balance \geqq \beta$ **then**
9: $SentList \leftarrow SentList \cup (NewSource \rightsquigarrow BestTarget, w, NewScore)$
10: **if** $(A + B) > MaxElimSupp$ **then**
11: $MaxElimSupp \leftarrow (A + B)$
12: $NewW \leftarrow NewW \cup w$
13: **if** $|NewSource| < MaxSource \wedge NewW \neq \emptyset$ **then**
14: **for** $x = AddSource + 1$ to $|SourceList|$ **do**
15: **if** $\forall S \in NewSource : (S \not\sqsubset SourceList[x])$ **then**
16: **if** $x < BlockNbr \times BlockSize$ **then**
17: $GrowSentinel(x, NewSource, NewBits, Target, NewW, NewScore)$
18: **else**
19: $SentQ \leftarrow SentQ \cup (x, NewSource, NewBits, Target, NewW, NewScore)$

better *Score* by combining it with one of the remaining filtered source measures in *SourceList*. For the remaining filtered source measures in *SourceList* that are not in a parent relationship with measures already included, and that are available (allocated), *GrowSentinel* will call itself to grow the sentinel further (lines 14 to 17). If a filtered source measure is not allocated, but otherwise qualified, the *GrowSentinel* will add its state to the queue, *SentQ*, and terminate (line 18 and 19). We note that meeting the thresholds is not a criteria for growing the sentinel further; as long as the *Score* is improving we have the potential for growing a sentinel that will eventually meet the thresholds. Also, we note that we will never add a source measure that occurred prior to *AddSource* in *SourceList*, this means that we are preserving the order of the source measures, and we are thus reducing the number of tested combinations.

Algorithm 7.5 *CalcScore*: Find best *Target* and calculate *Score*

CalcScore(bitmap *SourceBits*, bitmap *TargetBits*,
bitmap *ElimBits*, a measure *Target*, *SourceCnt*, *w*)

1: $A \leftarrow BitCount(TargetBits, \{PI\}) - BitCount(ElimBits, \{PI\})$
2: $B \leftarrow BitCount(TargetBits, \{NI\}) - BitCount(ElimBits, \{NI\})$
3: **if** $A \times B \geqq 0 \wedge |A + B| > 0$ **then**
4: **if** $A > 0 \vee B > 0$ **then**
5: $BestTarget \leftarrow Target$
6: **else**
7: $BestTarget \leftarrow \overline{Target}, A \leftarrow -A, B \leftarrow -B$
8: $SentSupp \leftarrow BitCount(SourceBits, \{PI, NI\})$
9: $Confidence \leftarrow \frac{|A|+|B|}{SentSupp}$
10: $Balance \leftarrow \frac{4 \times |A| \times |B|}{(|A|+|B|)^2}$
11: $NewScore \leftarrow (1 - wp + \frac{(1+Maxw-w) \times wp}{Maxw}) \times (\frac{1}{2} + \frac{1+MaxSource-SourceCnt}{MaxSource \times 2}) \times (A + B) \times Confidence \times (\frac{1}{2} + \frac{Balance}{2})$
12: **else**
13: $NewScore \leftarrow 0$
14: **return** *BestTarget*, *NewScore*, *SentSupp*, *Confidence*, *Balance*, A, B

The function *CalcScore* calculates the *Score* (Formula 7.10) for a potential sentinel (excluding division by the constant $MaxElimSupp$ as explained above). A sentinel can be represented by a bitmap resulting from the AND operation of all source measure bitmaps, *SourceBits*, and two bitmaps representing an AND opera-

tion of *SourceBits* and the bitmaps for a target measure, *TargetBits*, and the opposite of this target measure, *ElimBits*, respectively (Chapter 6). The term "opposite" is used deliberately since the target measure can also be inverse, thus the opposite in this case would be the original target measure. *CalcScore* (Algorithm 7.5) calls *BitCount* (elaborated in Chapter 6) to identify A and B (lines 1 to 2). We test to see that A and B have the same sign (line 3), because this is a prerequisite to fulfilling the criteria that $A \geqq 0 \wedge B \geqq 0$ from Formula 7.5 (A and B are defined in Formula 7.9). Furthermore, we test that at least one of them is different from zero since this is a prerequisite for getting a sentinel that complies with the thresholds. Subsequently we test whether the sign is negative or positive (line 4), this allows us decide whether $NewSource \rightsquigarrow Target$ or $NewSource \rightsquigarrow \overline{Target}$ is a valid sentinel (lines 4-7), since A and B are the same values but with opposite signs for these two sentinels. If $NewSource \rightsquigarrow \overline{Target}$ is the valid sentinel, the sign of A and B is changed to positive (line 7). Simply changing the sign of A and B when $\overline{Target}$ is the best target measure, *BestTarget*, allows us to *reduce the number of sentinels tested to almost half compared to prior art* (Chapter 6) where $NewSource \rightsquigarrow Target$ and $NewSource \rightsquigarrow \overline{Target}$ were tested separately. Once we have decided on the proper direction of *Target*, it is trivial to complete the calculations of *SentSupp* (Formula 7.6), *Confidence* (Formula 7.9), and *Balance* (Formula 7.8) in lines 8 to 10. These values are used to calculate *NewScore* (line 11) which is equal to Score (Formula 7.10) with the exception of the division by $MaxElimSupp(C, o, w)$, since it is constant and thus irrelevant to the ranking of sentinels. Moreover, $MaxElimSupp(C, o, w)$ is not known until we have inspected all sentinels in C. Therefore, the scores output for the sentinels mined are adjusted to comply with Formula 7.10 upon finalization of the *SentBMD* algorithm (Algorithm 7.1, line 8).

This concludes the walk-through of the *SentBMD* algorithm. In Section 7.7 we demonstrate that not only is *SentBMD* able to mine sentinels that cannot be found with prior art, it is also two to three times more efficient than prior art when mining regular sentinels.

7.6 Implementation

The *SentBMD* algorithm was implemented in C++ and compiled into a stand-alone 64-bit executable file using Microsoft Visual Studio 2010. We apply an Apriori-style optimization by pruning the measures that will never be able to be part of a conforming sentinel. Specifically, we know that in order for a source measure, or a target measure for a given w, to be part of a conforming sentinel, it needs to have at least σ indications in its indication sequence (to fulfill $SentSupp \geqq \sigma$). Therefore, we can simply dismiss any bitmap for a filtered source measure where $BitCount(Bitmap(SourceList[AddSource]), \{PI, NI\}) < \sigma$ by not appending it

from *SourceList* in Algorithm 7.4, lines 14 to 19. In multidimensional sentinel mining in particular we would expect that a large number of filtered measures can be pruned since we operate at a much finer granularity than for regular sentinels.

Based on prior art we have found that the most efficient way to count the bits set in a bitmap with our *BitCount* function is by exploiting the instruction known as *population count* (POPCNT) returns the number of bits set, and it is part of the SSE4a instruction set available, e.g., on the Intel "Core i7" and the AMD "10h" processor families (Chapter 6). In order to access this processor specific POPCNT instruction, we used a so-called *intrinsic* function.

With regards to parallelization, we set up a threshold for the number of invocations (one per core) of *EncodeBlock* that are spawned from *SentBMD* in parallel (Algorithm 7.1, line 9), and the remaining cores available are used for spawning the invocations of *GrowSentinel* (Algorithm 7.1, line 15). This threshold will allow us to control the balance between the resources used for loading and encoding, and the resources used for testing sentinels while the *EncodeBlock* invocations are running (See Section 7.7 for recommended setting). In this regard, we note that the algorithm does not require the blocks to arrive in sequence, and thus we can focus on creating workload balance between load/encode and sentinel test. In addition, the *GrowSentinel* invocations that are retrieved from the queue, *SentQ*, can vary in workload because some are closer to finalization than others, and thus we spawn these in groups of 5 to each core (rather than dedicate a core for each invocation) to ensure a proper workload for each spawn.

Computational Complexity: When we examine the *SentBMD* algorithm, we note that the worst case computational complexity will be dominated by the encoding of bitmaps in *EncodeBlock* (Algorithm 7.2), and the recursive mining process using *GrowSentinel* (Algorithm 7.4). All invocations of *EncodeBlock* can be performed in time $\mathcal{O}(n \times p \times q)$, where *n* is the number of periods, *p* is the number of source measures, and *q* is the total number of dimension values except for the time dimension values (*n*). The recursive mining process can be performed in time $\mathcal{O}(\frac{n}{WL} \times (p \times q)^l \times m)$, where *WL* is the CPU word length in bits, *l* is *MaxSource*, and *m* is *Maxw*. Therefore *SentBMD* has a worst case complexity of $\mathcal{O}(n + (\frac{n}{WL} \times (p \times q)^l \times m))$, since $(p \times q)^l >> (p \times q)$. By stressing this complexity as a worst case scenario, we recognize that the recursive growth based on *Score* is dependent on the data mined, and as seen in Section 7.7, the data can indeed influence the actual time consumption. Depending on whether $n >> (p \times q)^l \times m$ or not, the computational complexity will be dominated by either the encoding or the recursive mining process. We also note, that the parallelization will only alter the constant factors, and thus not the $\mathcal{O}$-complexity, since the number of CPU cores is a small constant. In Section 7.7 we test this assessment through extensive experiments.

7.7 Experiments

Setup: We run our experiments on an Intel Core i7 Quad CPU (920) 2.66GHz PC with 24GB RAM and 2 500GB disks (7,200 RPM) running a 64Bit version of Windows 2008 Server R2 (operating system) and a 64Bit version of Microsoft Analysis Services 2008 R2 (multidimensional database server). We use a range of synthetic datasets and a range of real-world datasets for the experiments. The synthetic datasets have 150 source measures and were created using three simple random walks such that 30 source measures have 80% chance of changing α, up or down, 70 source measures have 40% chance of changing α, up or down, and the remaining 50 source measures do not change above α. In our experience with real-world data, having a dataset where $\frac{1}{5}$ of the measures change many times, whereas $\frac{1}{3}$ of the measures have no relevant changes, is very realistic. The target measure was aligned such that it always changed one period later synchronized with the first source measure with many changes. This means that there will always exist at least one sentinel with $w = 1$ in these data. The synthetic datasets range from 100,000 to 1,000,000 rows in 100,000 row intervals. We note that the sizes of these datasets are *huge* compared to the real-world dataset. In general, we would expect any real application of sentinels to work on *significantly fewer rows* since we typically aggregate the data, e.g., into months or weeks, before mining sentinels. The real-world datasets are produced from the operational data warehouse of TARGIT A/S. Based on experience with more than 3,800 customers worldwide, we will characterize this dataset as typical for a medium-sized company with a mature data warehouse. The original dataset contains 241 months (20.1 years) of operational data across 148 source measures. We created descendants of the real-world dataset from 100,015 to 999,909 rows in $\approx 100,000$ row intervals by duplication. Descendants of the synthetic datasets are produced by selecting the source measure "tied" to the target measure, and the remaining number of source measures randomly to produce datasets with 10, 20, 30, ..., 150 source measures. Descendants of the real-world datasets are produced similarly, but with all source measures selected randomly.

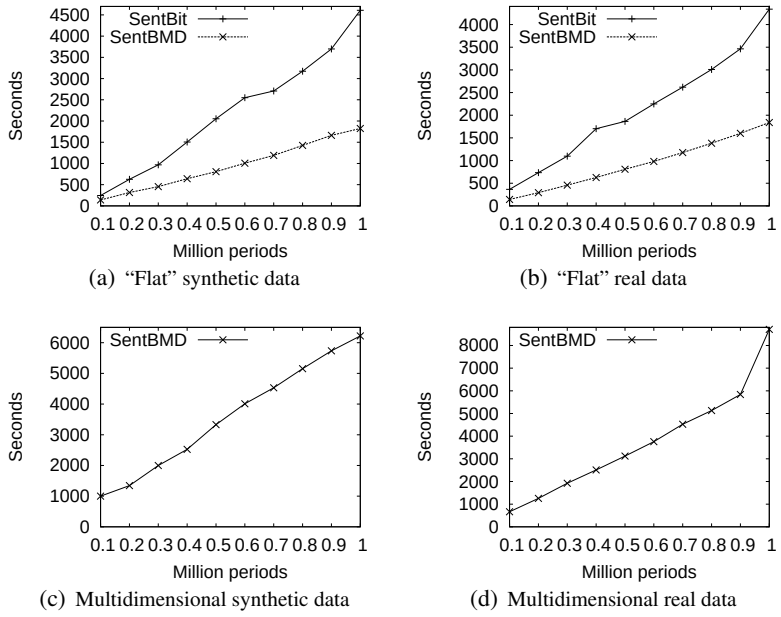
The multidimensional datasets have been creates by projecting two dimensions into the real and the synthetic datasets. The two dimensions have 11 ($\top$ + 10 values) and 21 ($\top$ + 20 values) values respectively. For each dimension combinations, a number of records from the respective source (synthetic or real) is appended in accordance with the desired size of the dimension. We have created the sizes of the dimensions such that they abide to a *power law*, specifically, the first member accounts for 20% of all rows in a given multidimensional dataset, second member accounts for 20% of the remaining rows (16% of the total), and so forth. The fact that power laws typically exists in real data has been observed in our previous work (Chapter 3), and to our best of knowledge this is the most realistic projection of the dimensions. In this

regard, it should be noted that the term "real data" is used about the dataset originating from TARGIT A/S although the dimensions have been synthetically induced in the multidimensional versions of this dataset. The multidimensional datasets range from 100,000 to 1,000,000 rows in 100,000 row intervals.

When nothing else is specified, we either use the 100,000 rows version of the real-world dataset with all 148 source measures, or the synthetic dataset with 100,000 rows with all 150 source measures. We use the following algorithm settings: $BlockSize = \frac{\#\text{ source measures}}{\#\text{ cores allocated to } EncodeBloch}$, *wp*=0.5, *MaxSource*=3 or 2 (3 for "flat" datasets, 2 for multidimensional datasets), and thresholds: *SentSupp*=5, *Conf*=0.8, *Balance*=0.8, and *Maxw*=60.

As mentioned in Section 7.6, *SentBMD* has been implemented with an option to control the balance between the number of cores used for spawning *EncodeBlock* and *GrowSentinel* respectively. In our experiments we set this balance to 50/50, i.e., 4 cores for spawning each of these invocation types while one or more *EncodeBlock* invocations are running.

Scaling dataset size: In Figure 7.4, we validate that *SentBMD* scales linearly to 1,000,000 periods of synthetic (Figure 7.4(a)) and real data (Figure 7.4(b)), and that it is more efficient than the *SentBit* algorithm from prior art. We note that scaling to



(a) "Flat" synthetic data (b) "Flat" real data

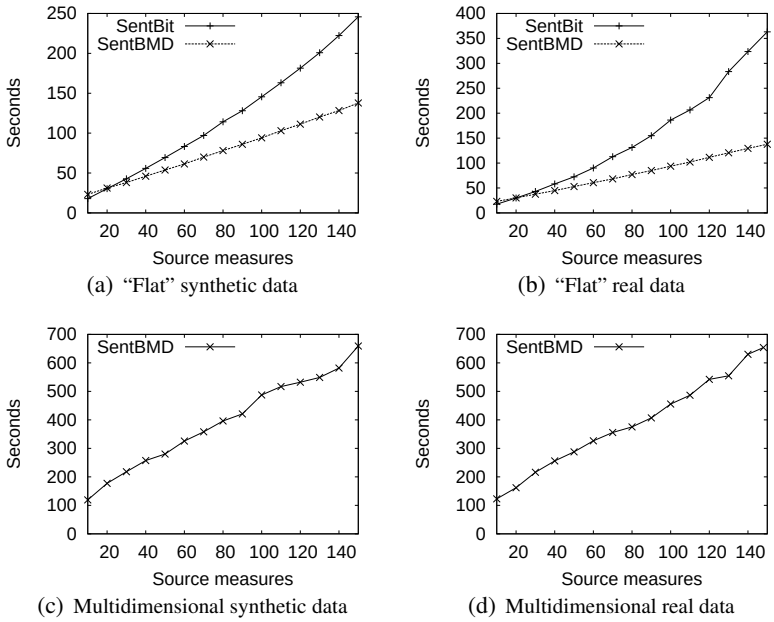
(c) Multidimensional synthetic data (d) Multidimensional real data

Figure 7.4: Scaling dataset size

1,000,000 periods with 150 measures means that we are operating on a data cube with 150,000,000 measure values. On the flat datasets in Figures 7.4(a) and 7.4(b), we observe an almost straight line when scaling the dataset size for *SentBMD*, whereas *SentBit* has slightly more variation in its linear behavior. We attribute this variation to the fact that the loading from the multidimensional database server represents a greater part of the total runtime for *SentBit* (see Figure 7.2), and as we will observe in other experiments, greater variations will occur whenever the runtime is dominated by (the loading from) the multidimensional database server. When running on both comparable flat datasets, we observe *SentBMD* to 2.5 times faster than *SentBit*. In Figure 7.4(c) we observe linear behavior for *SentBMD* on multidimensional data. Based on our analysis of computational complexity, we will expect *SentBMD* to scale linearly on multidimensional data since multidimensionality adds to the number of source measures which is constant during the experiments in Figure 7.4 (and the other factors are also kept constant). This also explains the comparing of runtime between the flat and the multidimensional datasets, where *MaxSource* is set to 3 and 2 respectively. For example, the runtime for *SentBMD* on flat synthetic data for 1,000,000 periods and 150 source measures is $\approx 2,000$ seconds, whereas on multidimensional data the runtime is $\approx 6,000$ seconds. With the two added dimensions, we have $(11 + 21 + (11 \times 21)) \times 150 = 263 \times 150 = 39,450$ source measures when running on the multidimensional data. With reference to Section 7.6, we note that increasing the number of source measures is expected to have a squared impact in the multidimensional experiments (*MaxSource*=2) and a cubic impact on the flat datasets (*MaxSource*=3) (see also the scaling of source measures below). Therefore, we would expect the runtime on a multidimensional dataset to be at least $1.75 = \frac{150^2 \times 263}{150^3}$ times higher compared to a flat dataset. As seen when comparing Figures 7.4(a) and 7.4(c), the runtime is ≈ 3 times longer on the multidimensional data, and since the number of cells loaded in both cases is equal, we attribute the remaining overhead to the fact that more sentinels are found and grown in the case where we have multidimensional data. This challenge is particularly evident in Figure 7.4(d) where we scale the multidimensional version of the real dataset. Here we experienced *SentBMD* running out of physical memory when scaling the number of periods beyond 900,000. The algorithm had 20GB RAM available, and thus we observe that the linear behavior ends as the operating systems start swapping the RAM to disk. The reason for only experiencing physical memory overflow on the multidimensional real data is attributed to the existence of more sentinels on the lower dimensional levels in the real data. To put this into perspective, we recall that a real dataset in a real environment, e.g., the TARGIT case, will be mined on months or weeks, meaning that the 900,000 periods that *can* fit into RAM is an extreme figure compared to the 241 periods (months) in the original dataset. In summary we can say that *SentBMD* scales linearly one both

flat and multidimensional data, and that it is significantly more efficient than *SentBit* from prior art.

Scaling source measures: In Figures 7.5(a) and 7.5(b) we scale the number of source measures from 1 to 150 on a flat synthetic dataset and from 10 to 148 on a flat real dataset. With reference to Section 7.6 we note that increasing the number of source measures is expected to have a cubic impact, based on our assessment of the computational complexity for *MaxSource*=3 (and the other factors are also kept constant), and we have verified in the statistical tool R (`www.r-project.org`) that the curves for *SentBit* are indeed cubic. However, when fitting the comparable figures for *SentBMD* in R, we found that these were scaling quadratically (while at a quick glance appearing to scale linearly). Similarly, when scaling *SendBMD* on the multidimensional data in Figures 7.5(c) and 7.5(d) while using *MaxSource*=2, we observe behavior close to linear which is also verified in R. We attribute this improvement by a factor n^{-1} in actual runtime complexity over prior art to the optimizations we have made in the *GrowSentinel* (Algorithm 7.4) and the *CalcScore* (Algorithm 7.5) functions (See Section 7.5). Aside from the significant difference in scalability, we observe *SentBMD* to be 1.7–2.5 times faster than *SentBit* when running on all measures.



(a) "Flat" synthetic data

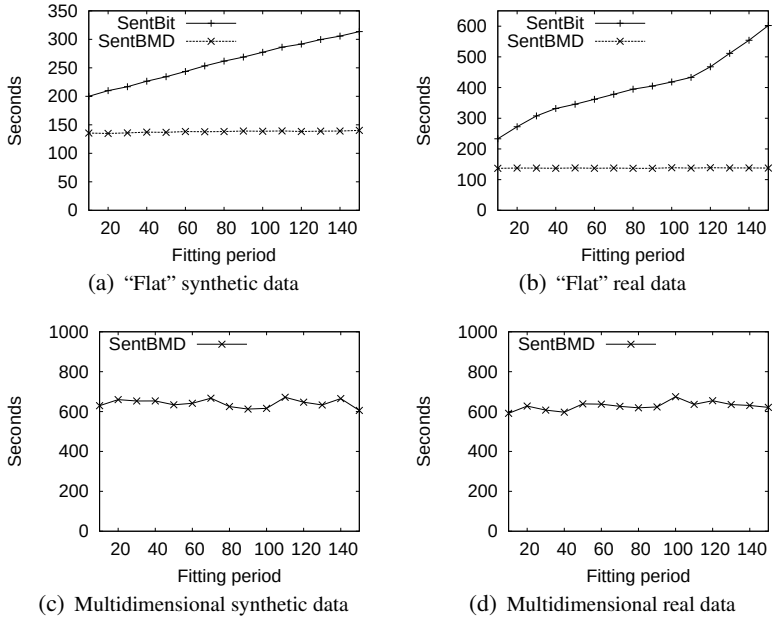
(b) "Flat" real data

(c) Multidimensional synthetic data

(d) Multidimensional real data

Figure 7.5: Scaling source measures

Scaling the fitting period: In Figure 7.6 we scale the number of periods over which we fit the warning periods for the sentinels. On the comparable flat synthetic (Figure 7.6(a)) and real (Figure 7.6(b)) datasets we observe *SentBMD* to run 2–4 times faster than *SentBit*. We also note that *SentBit* scales linearly (with a tendency not to scale linearly on extreme fitting periods) whereas *SentBMD* is constant when scaling the fitting period. We even see that *SentBMD* excels when the fitting period is scaled to the extreme. We attribute the high efficiency on larger fitting periods to the novel approach to testing warning periods in *SentBMD*. In *SentBMD*, the warning periods are "grown" along with the sentinel (See Algorithm 7.4) whereas SentBit uses a naïve approach and tests all possible warning periods. On the multidimensional data, Figures 7.6(c) and 7.6(d) we also observe the *SentBMD* algorithm to behave close to constant when scaling the fitting period.



(a) "Flat" synthetic data

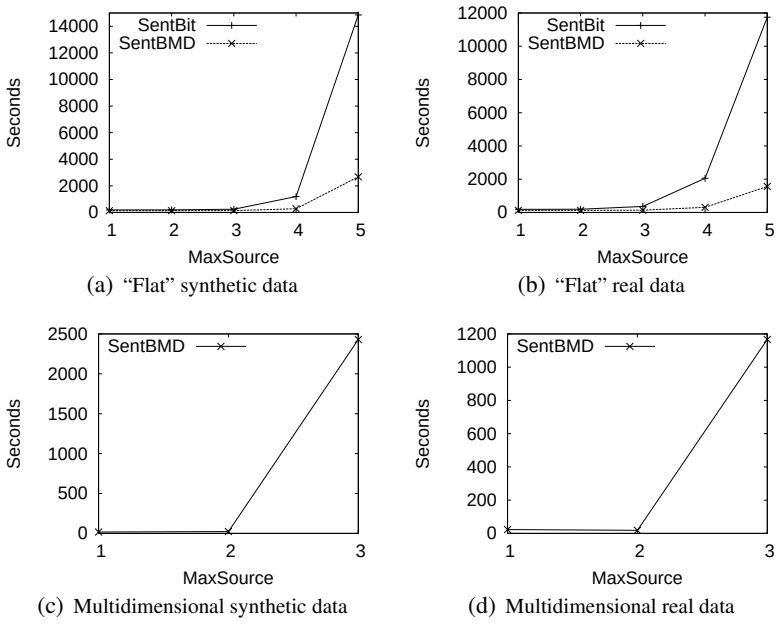
(b) "Flat" real data

(c) Multidimensional synthetic data

(d) Multidimensional real data

Figure 7.6: Scaling the fitting period

Scaling *MaxSource*: In Figure 7.7 we scale *MaxSource* for *SentBit* and *SentBMD* on the comparable flat synthetic (Figure 7.7(a)) and real (Figure 7.7(b)) datasets. As expected, we verified that the runtime for both algorithms grows exponentially in *MaxSource*. However, we note that *SentBMD* is faster by a factor of n^{-1} on the real data. This observation is in line with our previous observations when scaling the number of source measures. Scaling *MaxSource* on the multidimensional



(a) "Flat" synthetic data

(b) "Flat" real data

(c) Multidimensional synthetic data

(d) Multidimensional real data

Figure 7.7: Scaling *MaxSource*

data proved challenging since *SentBMD* ran out of memory on 100,000 periods for $MaxSource > 2$. This was due to a huge number of intermediate rules during the mining. We compensated by reducing dataset sizes to 1,000 periods, which allowed us to run *SentBMD* with $MaxSource \in \{1, 2, 3\}$ for synthetic (Figure 7.7(c)) and real (Figure 7.7(d)) data. Again, we observe the expected exponential behavior with reference to Section 7.6.

Experiments summary: In the experiments we observed the *SentBMD* algorithm to be significantly more efficient than the *SentBit* algorithm on comparable flat synthetic and real datasets. In addition, we observed *SentBMD* to meet our expectations with regards to our assessment of the worst case computational complexity. In the cases of scaling either the number of source measures or *MaxSource*, the *SentBMD* efficiency even beat our expectations by a factor of n^{-1} on the experiment data. In our experiments we have used data volumes that are much larger than one would expect in real world scenarios; and thus we believe that *SentBMD* in its current form will be able to conduct multidimensional sentinel mining in most cases arising from the real world. We did, however, run out of RAM in some experiments and as stated in Section 7.8 we suggest some approaches to address this issue for future work. Prior to conducting the experiments in this section, we conducted ex-

periments to fit the optimal *BlockSize*. Our findings showed that the optimal setting to be $BlockSize = \frac{\#\text{ source measures}}{\#\text{ cores allocated to } EncodeBloch}$ on the configuration used in these experiments, i.e., distributing the encoding evenly to the available cores. We also tested the latest Microsoft PowerPivot in a Microsoft SharePoint as an alternative data store for our experiments, but we found that using the Microsoft Analysis Server delivered 20% faster load times of data to our algorithms. The configuration in our experiments thus represents the most optimal conditions we have been able to identify.

7.8 Conclusion and Future Work

This chapter significantly extended prior work to exploit the hierarchical nature of multi-dimensional databases by allowing sentinels to be discovered for certain dimension values for certain source measures. The proposed *SentBMD* algorithm can mine novel *multidimensional* sentinels using bitmaps, and it allows source measures to be progressively input during the mining process. In addition, the *SentBMD* algorithm utilizes new CPU architectures optimally by parallelizing the work over multiple cores and taking advantage of novel CPU instructions. We demonstrated that *SentBMD* is significantly more efficient than prior art even when mining ordinary (non-multidimensional) sentinel rules. Finally, we demonstrated that *SentBMD* scales efficiently to very large multidimensional datasets. It has previously been demonstrated that sentinels can find strong and general rules that would not be found by sequential pattern mining or correlation techniques (Chapter 2), this obviously also holds for multidimensional sentinel mining.

For future work, a top-down approach trading approximation for improved efficiency and less memory consumption should be explored. Furthermore, exploring the opportunities for compressing the bitmaps of encoded measures could reduce the memory consumption for *SentBMD* when mining datasets over many periods.

Chapter 8

Summary of Conclusions and Future Research Directions

This chapter summarizes the conclusions and directions for future work presented in Chapters 2–7, and Appendix A.

8.1 Summary of Results

This thesis introduced the novel concept of so-called sentinels, that can be used to expand the window of opportunity for an organization to act based on changes in the environment in which it operates. Sentinels are mined on multidimensional data in three forms: First, *Regular sentinels* represent one source measure to one target measure relationships. Secondly, *Generalized sentinels* represent multiple source measures' relationships to one target measure. Third, *Multidimensional sentinels* represent generalized sentinels that hold for a subset of the data in a multidimensional data cube. Sentinels in all three forms were formally presented, including measures that can be used to assess the quality of a sentinel as well as for optimization and sorting purposes. Four different algorithms for sentinel mining were presented: *SQL-based* for regular sentinel discovery, *SentHiRPG* and *SentBit* for generalized sentinel discovery, and *SentBMD* for multidimensional sentinel discovery. Aside from expanding the capabilities of the algorithms, the work demonstrated a significant progression in the efficiency of sentinel mining, where the latest bitmapped algorithms, *SentBit* and *SentBMD*, are 3–4 orders of magnitude faster than the first SQL-based algorithm. This work also led to the industrial implementation of sentinel mining in the commercial software TARGIT BI Suite, which attracted the attention of leading industry analysts. In summary, the work in this thesis has demonstrated sentinel mining to be useful and unique. In the following, we go through each of the presented chapters and summarize the most important results.

Chapter 2 presented the first approach to discovering regular sentinels in a multi-dimensional database using an SQL-based algorithm. The chapter introduced the sentinel concept as a way to to expand the window of opportunity for an organization to act based on changes in the environment in which it operates. Formally, regular sentinels were defined at the schema level in a database, which means that they are more general than data level rules and thus are more general and cleansed for contradictions, and thus easy to interpret. In particular with regards to contradictions within a rule, this chapter presented an approach for eliminations of these. The chapter demonstrated a straight forward SQL-based algorithm for discovery of regular sentinels, and it was demonstrated that it scales linearly on large volumes of both synthetic and real-world data. From a qualitative perspective, it was demonstrated that sentinel rules with relevance for decision making can be extracted from real-world data. Furthermore, the possibility for automatic fitting of both warning and observation periods was highlighted. With regards to novelty, it was specifically demonstrated that sentinel rules are different from sequential pattern mining, since sentinel rules operate at the schema level and use a contradiction elimination process to generate fewer, more general rules. Furthermore, it was demonstrated that sentinel rules are complementary to correlation techniques, in that they could discover strong relationships between a smaller subset within a dataset; a relationship that would otherwise be "hidden in the average" using correlation techniques alone.

Chapter 3 proposed a significant generalization of the regular sentinels from the previous chapter. The chapter extended the concepts from Chapter 2 to allow multiple source measures to be combined in a relationship with a target measure. The chapter also introduced two novel quality measures *Balance* and *Score* that can assist in describing the degree of bi-directionality of a sentinel as well as the overall quality of a sentinel. The desirability of bi-directionality vs. uni-directionality was highlighted in the chapter. The algorithm, *SentHiRPG*, for generalized sentinel discovery was presented, which used *Score* for optimization in order to auto-fit the best warning period for the sentinel. In addition, the *SentHiRPG* algorithm was optimized with the introduction of a novel table of combinations (TC) and a reduced pattern growth (RPG) approach. Given a target measure, the *SentHiRPG* algorithm could autonomously find the best warning period and output the best sentinels from this. It was demonstrated that the basic version of *SentHiRPG* could produce more than ten times improvement in performance over the SQL-based algorithm. In addition, the effect of RPG was demonstrated by scaling the number of source measures where a behavior close to linearly was observed when combining large sets of source measures. Similarly, the hill-climbing optimization was demonstrated to be efficient when fitting the warning period over large period intervals. Finally, it was demonstrated that the *SentHiRPG* algorithm scales linearly on large volumes of both real and synthetic data.

Chapter 4 presented a demonstration of the user experience when mining sentinels in the TARGIT BI Suite version 2K10. It was demonstrated that sentinel mining could be done with few clicks in the TARGIT BI Suite, and thus little training and instruction is needed for a common user to be able to benefit from the technology. The reason for this usability is in part, that the underlying *SentHiRPG* algorithm is able to utilize the quality measure, *Score*, as well as optimizations from the previous chapter to autonomously assess the quality of the sentinels on behalf of the user. Another part of the user experience is defined by the so-called "few clicks" mindset, that surrounds any feature implemented in the TARGIT BI Suite. The demonstration thus highlighted the integrated experience, that resulted in sentinels being rated the most interesting and promising feature of TARGIT BI Suite version 2K9 in April 2009 by TARGIT partners representing a market footprint of 1,936 customers with more than 124,000 users. In addition, leading industry analyst, Gartner, introduced TARGIT in their Magic Quadrant for BI Platforms in 2010 and listed sentinels as one of the key strengths of TARGIT. Specifically, the demonstration in this chapter showed the dialogue flow where users, without any prior technical knowledge, are able to select a critical measure, a number of cubes, and a time dimension, and subsequently mine and schedule sentinels for *early warnings*. In addition, data warehouse setup that was a prerequisite for the demonstration was also presented.

Chapter 5 described the underlying technology that is presented in Chapter 4, and it describes the underlying implementation that has been done in the TARGIT BI Suite version 2K10 in order to provide the users with sentinel mining. The intuitive presentation of the sentinel concept from the previous chapter was elaborated and exemplified both a data and a user perspective. The architecture of the TARGIT BI Suite was described in detail, and the interfaces that are involved in sentinel mining were identified. Specifically, the dialogues between the TARGIT client and the TARGIT ANTserver were described. Furthermore, the TARGIT ANTserver component was described layer by layer. The *SentHiRPG* algorithm was also described in the context of the implementation. The feedback from both TARGIT customers as well as industry analysts was described, and the indications were that sentinels were useful and unique. In the context of the TARGIT BI Suite, sentinel mining was tested on a real-world operational data warehouse located in TARGIT A/S, and it was demonstrated through extensive experiments, that mining and usage of sentinels is feasible with good performance for the typical users on a real, operational data warehouse. Finally, a survey of competing companies in the business intelligence field in Gartner's Magic Quadrant 2010 was conducted, and the conclusion of this survey supported that sentinels are indeed unique in the market.

Chapter 6 demonstrated how the sentinel mining problem could be expressed as bitmap operations. The encoding of measure changes into bitmaps, so-called *indication streams*, was the prerequisite for mining sentinels using *indication joins* and

subsequently counting the number of bits set in the resulting bitmaps. The efficient algorithm for bitmapped sentinel mining, *SentBit*, had the ability to benefit from both parallelization as well as dedicated processor instructions for counting set bits. This means that *SentBit* is capable of benefitting from the multi-core architectures and instruction sets available on modern processors. The fully optimized *SentBit* algorithm proved to be 2–3 orders of magnitude faster than the *SentHiRPG* algorithm from Chapter 3, and thus 3–4 orders of magnitude faster than the SQL-based algorithm. Particularly in comparison to the *SentHiRPG* algorithm, it should be noted that *SentBit* does not use any approximation, and thus provides exact results unlike *SentHiRPG*. It was demonstrated through extensive experiments that that the *SentBit* algorithm scales scales efficiently to very large datasets of both real and synthetic data. Another important aspect in this chapter was the experiment supporting that sentinels are generally only found in a dataset if the data contains statistically significant relationships, this was demonstrated by creating a completely randomized dataset on which it was tested that sentinel mining did not find any rules as opposed to the other datasets mined.

Chapter 7 extended the formal definitions of generalized sentinels into multidimensional sentinels by introducing so-called *filtered measures*. Based on the formal definition, the chapter presented the *SentBMD* algorithm for multidimensional sentinel mining which significantly extended the bitmapped sentinel mining known described in previous chapter. The *SentBMD* algorithm was able to exploit the hierarchical nature of multi-dimensional databases and thus discover sentinels that hold for certain dimension values for certain source measures. Similarly to the *SentBit* algorithm, *SentBMD* was able to benefit from the novel CPU instructions available on modern processors. However, with regards to parallelization, *SentBMD* was significantly more advanced since it allowed source measures to be progressively input during the mining process. The progressive input meant that the encoding of source measures could be be done in parallel with the mining process of already encoded measures, and thus the *SentBMD* algorithm could utilize new multi-core CPU architectures efficiently. It was demonstrated that *SentBMD* was significantly more efficient than *SentBit* from previous chapter, even when mining ordinary (non-multidimensional) sentinel rules. In this context it should be noted that the *SentBMD* algorithm is also structured more efficiently then *SentBit* with regards to testing different measure relationships for sentinels, and thus *SentBMD* will also be more efficient even when there is no possibility to parallelize on a multi-core CPU. In general, it was demonstrated that the *SentBMD* scaled efficiently to very large multidimensional datasets of both real and synthetic data.

Appendix A explained in more detail how sentinels are linked to the CALM philosophy, and how it uses the Observation-Orientation-Decision-Action (OODA) con-

cept as a mean to identify three new desired technologies in business intelligence applications that improve the speed and quality in the decision making processes.

The thesis has thus formally presented a novel concept of sentinel mining in progressively more advanced forms, namely: regular sentinel mining, generalized sentinel mining, and multidimensional sentinel mining. The process of sentinel mining has also been demonstrated in more intuitive forms, such that non-technical readers will be able to comprehend the concept. The value of sentinel mining has been theoretically justified to benefit users in reacting faster whenever patterns occur that could indicate that a critical measure might be influenced within a given time frame. It has been demonstrated that the benefits of sentinels cannot be achieved by prior art such as association rule mining, sequential pattern mining, gradual rule mining, correlation, and regression techniques. In particular, the bi-directional property of sentinels is relevant, since it strengthens the support criterion of a rule relationship to make it reflect changes in two directions rather than only one. This also includes the elimination process that penalizes relationships that have conflicting support "inside the rule". Bi-directional sentinels are thus more likely to represent strong causal rule relationships between measures in a multidimensional database. Another aspect of sentinels is that they represent rules that hold for subsets of the data mined, this means that the rules can pick up relationships that would go undetected when using approaches such as regression or correlation that are efficient for identifying overall trends. In this context, we should recall that the primary objective for sentinels is to *warn* about possible influences on a critical measure, and thus sentinels are not intended to reveal an "universal truth". The ability to identify relationships that hold for smaller subsets is highly relevant in a real-world operational environment, since it is much more likely that these subset-relationships can be influenced by a user, as opposed to influencing global trends. The thought is that sentinels will create a much more *aware* type of organization that continuously seek to be proactive when interacting with the world around it.

The thesis provided four algorithms that each represent a step in the direction of the efficient bitmap-based *SentBMD* algorithm that can discover both generalized and multidimensional sentinels. Two of these algorithms have subsequently been applied in industry applications and deployed to real-world users. The latest TARGIT BI Suite version 2K10 SR1 uses the bitmapped algorithm described in Chapter 6. The industrial application of sentinels have been positively recognized by both customers and industrial analysts, and it seems that sentinel mining could finally deliver to the promise of end-user data mining that has been an industrial challenge for more than a decade in the business intelligence industry.

8.2 Research Directions

Several directions for future work remain for the work presented in this thesis. As mentioned in Chapter 7, an opportunity for improved efficiency and less memory consumption by applying approximation should be explored. The approximation should exploit the multi-dimensional environment in a top-down approach where only dimensions levels that are likely to be filters in good sentinels are selected. Such an approach could rely on the quality measures introduced in Chapter 3 in order to determine the improvement in quality when "expanding" a given dimension level. Using such an approach would allow the sentinel mining algorithm to autonomously determine the location and shape of the data areas used for filters. Such an approach could significantly reduce the number of filters that need to be tested, and thus it will have a good potential for significant performance improvements compared to the current exhaustive search in *SentBMD* where all filters are tested.

Another aspect that would be interesting to investigate is the memory consumption for *SentBMD* when mining datasets over many periods. In Chapter 7, we experienced that *SentBMD* memory consumption became an issue in multi-dimensional sentinel mining since the number of intermediate rules increases significantly. One strategy to overcome this issue could be to demand more in order to allow a sentinel to grow than simply an improved *Score*, e.g., setting a threshold for minimum *Score* during this process as well. Another strategy could be to compress the bitmaps that reside in memory by using run length encoding (RLE) or similar, e.g., PLWAH [16]. Other future work, in this context, could also seek to overcome the memory consumption issue by running sentinel mining on systems optimized for optimal usage of external memory, e.g., systems that combine RAM and solid state drives (SSD) as opposed to RAM and disk.

With regards to the capabilities of the sentinel mining algorithms in this thesis, there is an opportunity for future work to extend the algorithms to allow for more target measures to be mined in parallel, perhaps even to consider all measures to be both source and target measures. In such an algorithm, all relationships could be mined and subsequently navigated for greater organizational insight in addition to the warning capabilities of the existing sentinels. Another way to extend the sentinel mining capabilities could be to allow the warning period to be less strict, e.g., an interval, "less than," or "greater than."

Aside from seeking more efficiency in the mining process, it would also be interesting to conduct a qualitative study in which the differences in relevant results from sentinel mining and other techniques are compared and evaluated, e.g., sequential pattern mining, gradual rule mining, bellwether analysis, and various correlation techniques.

Finally, as suggested in Chapter 5, the ability to visualize the sentinels found could be explored. For future organizational success with sentinels, there is a need for users to understand the meaning of the sentinels found. Such an understanding is likely to arise from an interactive ability to visualize sentinels. In this context, a study of actual sentinel usage in the real world would be valuable to see if sentinel mining do indeed deliver to the promise of valuable end-user data mining.

Bibliography

[1] Advanced Micro Devices. *Software Optimization Guide for AMD Family 10h Processors*, November 2008.

[2] R. Agrawal, T. Imielinski, and A. Swami. Mining association rules between sets of items in large databases. *In Proc. of ACM SIGMOD*, pp. 207–216, 1993.

[3] R. Agrawal and R. Srikant. Fast Algorithms for Mining Association Rules in Large Databases. *In Proc. of VLDB*, pp. 487–499, 1994.

[4] R. Agrawal, K.I. Lin, H.S. Sawhney, and K. Shim. Fast similarity search in the presence of noise, scaling, and translation in timeseries databases. *In Proc. of VLDB*, pp. 490–501, 1995.

[5] R. Agrawal and R. Srikant. Mining Sequential Patterns. *In Proc. of ICDE*, pp. 3–14, 1995.

[6] R. Agrawal and J.C. Shafer. Parallel mining of association rules. *IEEE TKDE 8(6)*: 962–969, 1996.

[7] S. Ayouni, A. Laurent, S.B. Yahia, and P. Poncelet. Mining Closed Gradual Patterns. *In Proc. of ICAISC*, Part 1, pp. 267–274, 2010.

[8] C. Ballard, J. Rollins, J. Ramos, A. Perkins, R. Hale, A. Dorneich, E.C. Milner, and J. Chodagam. *Dynamic Warehousing: Data Mining Made Easy.* `ibm.com/redbooks`, September 2007.

[9] C. Bettini, X.S. Wang, and S. Jajodia. Mining temporal relationships with multiple granularities in time sequences. *IEEE DEBU 21(1)*: 32–38, 1998.

[10] P. Bosc, O. Pivert, and L. Ughetto. On Data Summaries Based on Gradual Rules. *In Proc. of Fuzzy Days*, pp. 512–521, 1999.

[11] S. Brin, R. Motwani, J.D. Ullman, and S. Tsur. Dynamic Itemset Counting and Implication Rules for Market Basket Data. *In Proc. of ACM SIGMOD*, pp. 255–264, 1997.

[12] T. Calders and B. Goethals. Mining All Non-derivable Frequent Itemsets. *In Proc. of PKDD*, pp. 74–85, 2002.

[13] B.C. Chen, R. Ramakrishnan, J.W. Shavlik, and P. Tamma. Bellwether Analysis: Predicting Global Aggregates from Local Regions. *In Proc. of VLDB*, pp. 655–666, 2006.

[14] B.C. Chen, R. Ramakrishnan, J.W. Shavlik, and P. Tamma. Bellwether analysis: Searching for cost-effective query-defined predictors in large databases. *ACM TKDD 3(1)*: Article 5, 2009.

[15] M. Csikszentmihalyi. *Flow: The Psychology of Optimal Experience.* Harper & Row Publishers, NY, 1990.

[16] F. Deliège and T.B. Pedersen. Position list word aligned hybrid: optimizing space and performance for compressed bitmaps. *In Proc. of EDBT*: pp. 228–239, 2010.

[17] T. Eavis, G. Dimitrov, I. Dimitrov, D. Cueva, A. Lopez, and A. Taleb. Parallel OLAP with the Sidera server. *Future Generation Computer Systems 26(2)*: pp. 259–266, 2010.

[18] T. Eavis. Parallel and Distributed Data Warehouses. *Encyclopedia of Database Systems*, pp. 2012–2018, 2009.

[19] T. Eavis and X. Zheng. Multi-level Frequent Pattern Mining. *In Proc. of DASFAA*, pp. 369–383, 2009.

[20] L. Feng, J.X. Yu, H. Lu, and J. Han. A template model for multidimensional inter-transactional association rules. *VLDB J. 11(2)*: 153–175, 2002.

[21] M. Garofalakis, R. Rastogi, and K. Shim. Spirit: Sequential pattern mining with regular expression constraints. *In Proc. of VLDB*, pp. 223–234, 1999.

[22] G. Graefe, R. Bunker, and S. Cooper. Hash Joins and Hash Teams in Microsoft SQL Server. *In Proc. of VLDB*, pp. 86–97, 1998.

[23] J. Han, G. Dong, and Y. Yin. Efficient mining of partial periodic patterns in time series database. *In Proc. of ICDE*, pp. 106–115, 1999.

[24] J. Han, J. Pei, B. Mortazavi-Asl, Q. Chen, U. Dayal, and M. Hsu. FreeSpan: frequent pattern-projected sequential pattern mining. *In Proc. of KDD*, pp. 355–359, 2000.

[25] J. Han, J. Pei, and Y. Yin. Mining Frequent Patterns without Candidate Generation. *In Proc. of ACM SIGMOD*, pp. 1–12, 2000.

[26] J. Han and M. Kamber. *Data Mining Concepts and Techniques.* (2nd Ed.) Morgan Kaufmann Publishers, 2006.

[27] E. Hüllermeier. Association Rules for Expressing Gradual Dependencies. *In Proc. of PKDD*, pp. 200–211, 2002.

[28] Information Builders. *Predictive Modeling: WebFOCUS RStat.* `informationbuilders.com/products/webfocus/PredictiveModeling.html`, current as of June 18th, 2010.

[29] Intel. *Intel SSE4 Programming Reference*, July 2007.

[30] R. Kimball. *The Data Warehouse Lifecycle Toolkit: Expert Methods for Designing, Developing, and Deploying Data Warehouses.* Wiley, 1998.

[31] A. Laurent, B. Négrevergne, N. Sicard, and A. Termier. PGP-mc: Towards a Multicore Parallel Approach for Mining Gradual Patterns. *In Proc. of DASFAA*, pp. 78–84, 2010.

[32] W.S. Lind. *Maneuver Warfare Handbook.* Westview Press, 1985.

[33] A.M. Lukatskii and D.V. Shapot. Problems in multilinear programming. *Computational Mathmatics and Mathmatical Physics*, 41(5):638648, 2001.

[34] Microsoft Corporation. *Data Mining Algorithms (Analysis Services - Data Mining).* `technet.microsoft.com/en-us/library/ms175595.aspx`, current as of June 18th, 2010.

[35] MicroStrategy Corporation. *MicroStrategy Data Mining Services.* `microstrategy.com/Software/Products/Service_Modules/DataMining_Services`, current as of June 18th, 2010.

[36] M. Middelfart. *CALM: Computer Aided Leadership & Management.* iUniverse, 2005.

[37] M. Middelfart. Improving Business Intelligence Speed and Quality through the OODA Concept. *In Proc. of DOLAP*, pp. 97–98, 2007.

[38] M. Middelfart and T.B. Pedersen. *Discovering Sentinel Rules for Business Intelligence.* DB Tech Report no. 24, dbtr.cs.aau.dk

[39] M. Middelfart and T.B. Pedersen. Discovering Sentinel Rules for Business Intelligence. *In Proc. of DEXA*, pp. 592–602, 2009.

[40] M. Middelfart, T.B. Pedersen, and J. Krogsgaard. Efficient Discovery of Generalized Sentinel Rules. *In Proc. of DEXA*, II, pp. 32–48, 2010.

[41] M. Middelfart and T.B. Pedersen. Using Sentinel Technology in the TARGIT BI Suite. *PVLDB 3(2)*: 1629–1632, 2010.

[42] M. Middelfart and T.B. Pedersen. Implementing Sentinel Technology in the TARGIT BI Suite. *In submission*.

[43] M. Middelfart, T.B. Pedersen, and J. Krogsgaard. Efficient Sentinel Mining Using Bitmaps on Modern Processors. *In submission*.

[44] M. Middelfart, T.B. Pedersen, and J. Krogsgaard. Multidimensional Sentinel Mining Using Bitmaps. *In submission*.

[45] T. Mitchell. *Machine Learning.* McGraw-Hill, 1997.

[46] F. Nakagaito, T. Ozaki, and T. Ohkawa. Discovery of Quantitative Sequential Patterns from Event Sequences. *In Proc. of ICDM*, pp. 31–36, 2009.

[47] Oracle Corporation. *Oracle Data Mining Mining Techniques and Algorithms.* oracle.com/technology/products/bi/odm/odm_techniques_algorithms.html, current as of June 18th, 2010.

[48] J. Pei, J. Han, B. Mortazavi-Asl, H. Pinto, Q. Chen, U. Dayal, and M.C. Hsu. PrefixSpan: Mining Sequential Patterns by Prefix-Projected Growth. *In Proc. of ICDE*, pp. 215–224, 2001.

[49] J. Pei, J. Han, B. Mortazavi-Asl, J. Wang, H. Pinto, Q. Chen, U. Dayal, and M. Hsu. Mining Sequential Patterns by Pattern-Growth: The PrefixSpan Approach. *IEEE TKDE 16(11)*: 1424–1440, 2004.

[50] N. Pendse and C. Bange. *The missing "Next Big Things".* olapreport.com/Faileddozen.htm, current as of June 16th, 2010.

[51] H. Pinto, J. Han, J. Pei, K. Wang, Q. Chen, and U. Dayal. Multi-Dimensional Sequential Pattern Mining. *In Proc. of CIKM*, pp. 81–88, 2001.

[52] M. Plantevit, A. Laurent, D. Laurent, M. Teisseire, and Y.W. Choong. Mining multidimensional and multilevel sequential patterns. *ACM TKDD 4(1)*: Article 4, 2010.

[53] C. Raïssi, T. Calders, and P. Poncelet. Mining conjunctive sequential patterns. *Data Mining and Knowledge Discovery 17(1)*: 77–93, 2008.

[54] R.L. Sallam, B. Hostmann, J. Richardson, and A. Bitterer. *Magic Quadrant for Business Intelligence Platforms.* `gartner.com/technology/media-products/reprints/oracle/article121/article121.html`, current as of June 18th, 2010.

[55] S. Sarawagi. Explaining Differences in Multidimensional Aggregates. *In Proc. of VLDB*, pp. 42–53, 1999.

[56] SAS Corporation. *Data mining with SAS Enterprise Miner.* `sas.com/technologies/analytics/datamining/miner`, current as of June 18th, 2010.

[57] R. Srikant and R. Agrawal. Mining Sequential Patterns: Generalizations and Performance Improvements. *In Proc. of EDBT*, pp. 3–17, 1996.

[58] T. Wu, Y. Chen, and J. Han. Association Mining in Large Databases: A Re-examination of Its Measures. *In Proc. of PKDD*, pp. 621–628, 2007.

[59] J. Yang, W. Wang, P.S. Yu, and J. Han. Mining long sequential patterns in a noisy environment. *In Proc. of ACM SIGMOD*, pp. 406–417, 2002.

[60] Y. Zhu and D. Shasha. StatStream: Statistical Monitoring of Thousands of Data Streams in Real Time. *In Proc. of VLDB*, pp. 358–369, 2002.

[61] R. Zurawski (ed.), C.S. Jensen, and T.B. Pedersen. *The Industrial Information Technology Handbook*, Multidimensional Databases and OLAP. CRC Press, 2005.

Appendix A

Improving Business Intelligence Speed and Quality through the OODA Concept

For the past few decades, a lot of focus has been put on the immediate data warehousing challenges of obtaining and storing the data needed for business purposes, in a manner so that it can be accessed with a reasonable performance [61], and although a number of applications have emerged in the OLAP space, there has been little theorizing about the management challenge of turning stored data into meaningful and valuable decisions, with perhaps the exception of Kimball's "Analytical Cycle" that suggests a specific way to conduct a business analysis [30].

The framework proposed as part of the Computer Aided Leadership & Management (CALM) thesis [36] can assist with a heuristic approach to applying the technologies available at any given time, to the challenges faced by leaders and managers; moreover this framework can assist in uncovering new areas for future research that will benefit organizations in the years to come.

The idea is to take the Observation-Orientation-Decision-Action (OODA) loop, which was originally pioneered by Top Gun fighter pilot John Boyd in the 1950s [32], and group and categorize the available business intelligence technologies according to their role in the four processes in the OODA loop, namely:

Observation - in this phase we use technology to look at the data with an expectation of what it should be, e.g. through dashboards, reports and agents.

Orientation - in this phase we look at the data in different ways depending on what it shows, e.g. through analysis, data mining and simulation. Typically this phase is initiated after something in the observation phase has proven to be different from what we expected.

Decision - this phase is currently undertaken primarily by human intelligence, but the results found in the Data Warehouse can be evaluated against other external or internal -typically unstructured- data.

Action - based on the decision made, we seek to implement the course of action chosen, either through the IT infrastructure as a business process in a Service Oriented Architecture or as communication such as blog, email, phone, or plain face-to-face communication. Any organization and its purpose can be described as a number of OODA loops that are continuously cycled by users and conform to one or more Key Performance Indicator's (KPI's) that define their success. However, it should be noted that the more OODA loops that exists, the more the organization is continuously adapting to its environment; and the faster the users are able to cycle through the OODA loop, the more competitive the organization will be in correcting problems and seizing opportunities to improve performance.

If we accept the business benefits for an organization of its users cycling through multiple OODA loops with as high speed as possible, we need to perform research into technologies that bridge all the technologies mentioned in the OODA technology breakdown above. Based on this, the suggestion for three new research directions in the business intelligence area is:

First, technologies that reduce the number of user interactions needed to cycle through an OODA loop. Perhaps we can learn from the specialist systems already existing in airline autopilots and Anti-lock Brake System (ABS) to create smart general systems, that in addition to be operated, can be customized by business users to allow computers to significantly reduce the number of human interactions, e.g. clicks, and thus the time spent cycling the OODA loop. In some cases such systems would ideally operate autonomously, and thereby allow users to focus on processes where computers are less advantageous. This would create true human/computer synergy rather than simple collaboration.

Secondly, having established a number of OODA loops with a number of KPI's assigned, we might provide the users with technologies that can help them identify patterns that can act as "sentinels". These sentinels, causal or not, are ideally measures that can give early warnings about a later influence on a business critical measure. One can intuitively imagine that the number of actual sales visits at a given time might impact the revenue at some later time, or perhaps the company's footprint on a number of blogs might indicate something about product quality. The aim here should be to provide the business users with technologies that automatically identify measures that qualify for sentinels in order to find the best, perhaps less intuitive sentinels.

Finally, we could consider running Business Process Intelligence on the entire system of OODA loops. If we measure the time spent in an OODA cycle, and we can assess the "quality" in the sense of our ability to conform to the KPI's assigned to

the OODA loop, we can begin to rank the OODA loops depending on their speed and quality. If for instance an OODA loop is improving in both speed and quality, we have most likely automatically identified an area in which the organization experiences "Flow" [15], in other words, the organization is doing something better and better with less and less effort. On the other hand, if an OODA loop is not improving in terms of speed, quality or both, the management can easily identify this area. On a broader scale, such an OODA assessment of an organization might be a valuable quantitative tool for talent and core competency management, and it might give upper management valuable information on where to invest or to divest. What is needed to give this kind of strategic advantage is technology, that can assess speed and quality of OODA loops in a general purpose business intelligence solution, but to get to a point where we can measure the OODA performance, we first need to manage our business intelligence technologies according to the OODA concept.

Hopefully, these three examples have illustrated the power of the OODA concept, which covers the idea to combine OODA loops with KPI's to render an organization agile and competitive using business intelligence. As new technologies are emerging, they can easily be categorized and evaluated for their business impact by measuring their effect on an OODA loop's speed and quality, and furthermore this framework can assist us identifying new areas for research simply by looking at the gaps and bottlenecks in the OODA loops.

Appendix B

Summary in Danish / Dansk resumé

Denne afhandling introducerer et nyt koncept: sentinel-regler (sentinels). Sentinels repræsenterer sammenhænge i data, typisk mellem data i en virksomheds ydre miljø og virksomhedens forretningskritiske processer. Formålet med sentinels er at advare forretningsbrugere om potentielle forandringer i deres nøgletal (Key Performance Indicators eller KPI'er) og derved gøre det muligt at korrigere deres handlinger, før ændringerne bliver en realitet. Ideen bag sentinels er beskrevet i Computer Aided Leadership & Management (CALM) filosofien, der omhandler en måde, hvorpå man i en virksomhed kan opnå en højere beslutningshastighed i en kaotisk verden. I denne afhandling er sentinels blevet til virkelighed, og således kan almindelige brugere på alle niveauer i organisationen drage nytte af advarsler fra sentinels.

Helt konkret er sentinels regelforhold på skemaniveau i en multidimensionel datakube. Disse forhold repræsenterer forandringer over tid på bestemte målepunkter, som efterfølges af en forandring i et andet vigtigt brugerdefineret målepunkt, typisk en KPI. En vigtig egenskab ved sentinels er bidirektionalitet, hvilket betyder, at forandringsforholdet også gælder for den komplementære retning. Dette betyder nemlig, at en bidirektionel sentinel har større chance for at være kausal end at være tilfældig. Sentinels kan variere i kompleksitet afhængig af antallet af målepunkter, der er inkluderet i reglen: Almindelige sentinels repræsenterer forhold, hvor ændringer i ét målepunkt fører til forandringer i et andet inden for en given tidsperiode. Generaliserede sentinels repræsenterer forhold mellem forandringer på flere målepunkter, som fører til forandringer på et andet målepunkt inden for en given tidsramme. Multidimensionelle sentinels kombinerer skema- og dataniveauerne, hvilket betyder, at hver målepunktsforandring i reglen kan gælde for enten hele eller dele af kuben. En generaliseret sentinel kan f.eks. advisere brugere om, at omsætningen sandsynligvis vil falde inden for to måneder, hvis der sker en forøgelse af de problemer, kunderne

oplever, kombineret med, at man observerer en nedgang i webtrafikken. En multidimensionel sentinel kan derimod advare disse brugere om, at omsætningen sandsynligvis vil falde inden for to måneder, hvis der sker en forøgelse af kundeklager i USA (såkaldt drilldown på den geografiske dimension), kombineret med, at man observerer en nedgang i det beløb, som investeres i kundesupport til bærbare computere (drilldown på produktdimentionen).

Det arbejde, som ledte frem til denne afhandling, udviklede sig fra algoritmer til opdagelse af almindelige sentinels, over algoritmer til opdagelse af generaliserede sentinels og endte med algoritmer til opdagelse af multidimensionelle sentinels med flere kilder og målepunkter. Derudover blev algoritmerne i stand til automatisk at finde de bedste advarselsperioder til en given sentinel. Udover at udvide algoritmernes kapabilitet, så skete der også en betydelig forøgelse af effektiviteten under opdagelsen af sentinels. De nyeste bitmap-baserede algoritmer, som også drager nytte af moderne CPU'er, er 3–4 størrelsesordner hurtigere end den første SQL-baserede sentinel-algoritme. Dette arbejde ledte også til implementering af sentinels i forretningssoftwaren TARGIT BI Suite, hvilket har fanget førende brancheanalytikeres interesse. Kort sagt har arbejdet i denne afhandling udviklet opdagelse af sentinels fra teoretisk idé til konkrete, effektive algoritmer, og arbejdet har ydermere vist, at sentinels er både nyttige og unikke.